The Design and Implementation of a Two and Three-Dimensional Triangular Irregular Network based GIS

Volume I

Alias Abdul-Rahman

July, 2000

A thesis submitted for the degree of Doctor of Philosophy

UNIVERSITY
of
GLASGOW

Department of Geography & Topographic Science
University of Glasgow

If the sea were ink
For the words of my Lord,
the sea would be spent
before the Words of my Lord are spent.

Qur'an. The Cave 18:109

In Loving Memory of My Mother
Kalthom Ismail
1985

Abstract

Abdul-Rahman, A. (2000). The design and implementation of a two and three-dimensional triangular irregular network based GIS. PhD Thesis

Keywords: spatial data (2D, 3D), data representations, object-oriented (OO) or object orientation, distance transformation (DT), Voronoi tessellations, Delaunay triangulation, triangular irregular network (2D TIN), 3D TIN or tetrahedral network (TEN), algorithms, data structures, data modelling, database, GIS (2D, 3D), OO GIS, TIN-based GIS, display user interface.

It has long been realised in the GIS community that most 2D GISs are capable of handling 2D spatial data efficiently, but systems have had less success with 3D spatial data. This is reflected in the current GIS market place where systems which can handle 3D data are hardly available - due to several impediments in implementing such systems. This thesis attempts to address some of the impediments. The impediments which relate to spatial data especially data representation, data structuring and data modelling using object-oriented (OO) techniques are the foci of this thesis. OO techniques are utilized because they offer several advantages over the traditional (i.e. structural) techniques in software development. In the aspect of spatial representation, several major representations are investigated, which then lead to identifying an appropriate representation both for 2D and 3D data, that is triangular irregular network (TIN) data structures. 2D data is represented by a 2D TIN, and 3D data is represented by a 3D TIN (also called a tetrahedral network or TEN). Several algorithms were developed for the construction of the data structures where procedures such as distance transformation (DT) and Voronoi tessellations were utilized. Besides standard Delaunay triangulations, constrained triangulations were also developed, thus the inclusion of real world objects in the spatial data modelling can be facilitated. Four classes of real world objects are identified (i.e., point, line, surface, and solid objects). For the purpose

of spatial data modelling of the four types of objects, a formal data structure (FDS) is utilized. An OO database development is also investigated. This is done via a commercial system called POET OO DBMS where spatial data query and retrieval can be performed. Further, two application programs are developed, namely contouring and volume computation. All the developed algorithms and methods were tested using real data sets. To facilitate the output of the developed methods and algorithms, software called *TinSoft*, which is a Windows-based Multiple Document Interface software was developed in this research.

Preface

This thesis is about the design and implementation of GIS using two types of data structures, namely 2D TIN and 3D TIN (or TEN). This research work is in two volumes, Volume I covers eight chapters reports on the research, while Volume II present the software code developed in this work.

The relationships of this work with the existing knowledge in the universal GIS domain may be summarised as follows (i.e. in terms of originality and derivation). Ideas which came from literature are categorised as 'derived', whereas 'original' indicates the contribution of the author.

Chapter 1 Introduction

1.1 Introduction	Derived and Original
1.2 Background to the Research Problem	Derived and Original
1.3 Research Objectives	Original
1.4 Research Scope	Original
1.5 Research Approach and Methodology	Derived and Original
1.6 Structure of the Thesis	Original
1.7 Commercial Tools	Original
1.8 Summary	Original

Chapter 2 An Overview of 3D GIS Development

2.0 Introduction	Derived and Original
2.1 GIS Functions	Derived
2.2 3D GIS	Derived and Original
2.3 Who needs 3D GIS	Derived and Original
2.4 Recent Progress Made on 3D GIS	Derived and Original
2.5 Commercial Systems Towards 3D GIS	Derived
2.6 Why is 3D GIS Difficult to Realise?	Original
2.7 Discussion	Original

Chapter 3 2D and 3D Spatial Data Representations

3.1 Introduction	Derived and Original
3.2 Classes of Object Representations	Derived and Original
3.3 GIS Applicability of the Representations	Original
3.4 The Selection Criteria	Original
3.5 Vector and Raster Representation	Original
3.6 Summary	Original

Chapter 4 Fundamental Aspect of Spatial Data Modelling and GIS

4.1 Introduction	Derived and Original
4.2 Spatial Data	Original
4.3 Spatial Data Modelling	Derived and Original
4.4 Spatial Data Structuring	Original
4.5 Relational Database Model	Derived and Original
4.6 Object-Oriented Database Model	Derived and Original
4.7 Object-Oriented Subsystems for GIS	Original
4.8 Database Management Systems (DBMS)	Derived
4.9 Geographic Information Systems (GIS)	Derived and Original
4.10 The OO TIN GIS	Original
4.11 Summary	Original

Chapter 5 Object Orientation of TIN Spatial Data

5.1 Introduction	Derived and Original
5.2 Object-Oriented Concepts	Derived and Original
5.3 Object-Oriented TIN Tessellations	Original
5.4 Object-oriented TINs Spatial Data Modelling	Original
5.5 Object-oriented TIN Spatial Database Dev.	Derived and Original
5.6 Object-Oriented TIN-based Subsystems for GIS	Original
5.7 Summary	Original

Chapter 6 The Supporting Algorithms

6.1 Introduction	Derived and Original
6.2 Distance Transformation	Original
6.3 Voronoi Tessellations	Original
6.4 Triangulations	Original
6.5 Visualization	Original
6.6 3D Distance Transformation	Original
6.7 3D Voronoi Tessellations	Original
6.8 Tetrahedron Network (TEN) Generation	Original
6.9 Constrained Triangulations	Original
6.10 Contouring Algorithm	Original
6.11 Summary	Original

Chapter 7 The Implementation and Test

7.1 The Study Area	Original
7.2 The Interface	Original
7.3 The Triangulations: 2D TIN and 3D TIN	Original
7.4 The Applications	Derived and Original
7.5 Height Interpolation for 3D Objects	Original
7.6 Input and Output File Formats	Original
7.7 Discussion	Original

Chapter 8 Conclusions and Future Research

8.1 Summary	Original
8.2 Discussion	Original
8.3 Conclusions	Original
8.4 Future Research	Original

Volume II (for entire sections) Original

Acknowledgements

I would like to express my sincere thanks especially to the supervision of Dr. Jane Drummond who always in position to guide and give advice to the right direction in pursuing this research work. Contribution from John Shearer is also gratefully acknowledged especially at the early stage of the work. I must admit that the knowledge gained from Dr. Ron Poet of the Computer Science Department, Glasgow University on the subject of object-oriented (OO) programming was very invaluable. He also helped generously on aspects of C++ programming in the first and second year. Technical advice received from Prasad Jeevanigi of POET.com USA was very helpful. Hints and advice from Ian Spencer who authored the book called Teach Yourself OWL Programming in 21 days was also great. To a former colleague at ITC (The Netherlands), Dr. Morakot Pilouk who is now in ESRI California with whom I communicated at the early stage of the research especially on the aspect of software development is gratefully acknowledged.

To other people at the Topographic Science Section of the Geography and Topographic Science Department such as Anne Dunlop who supplied field survey data sets, Brian Black and Stephen McGinley who were always there whenever my computer got trouble are also acknowledged for their excellent IT support. Also, to secretary Amy McNeill for her excellent job at the department's main office.

Thanks to my current employer, the Universiti Teknologi Malaysia (UTM) who provides the study leave and all the related fund to carry out this research work in the University of Glasgow. I also would like to thank my colleague and also friend, Assoc. Prof Ghazali Desa (my current Head, Department of Geoinformatics at UTM) for his support and encouragement. The same level of support received from Prof. Dr. Ayob Sharif (Dean of Faculty) is also acknowledged.

Again, to Dr. Jane Drummond, I greatly owed her marvellous and kindness in all aspect of life (socially and academically). This in fact, no doubt gives a tremendous strength to me as well as to my family. Her support for one week accommodation near Loch Alsh, north-west Scotland in the first year data collection exercise was brilliant. Various supports from her pushed away the misery of a research period and brought a wee bit of sunshine to our years of wet Glasgow life.

An understanding and continuos support from my family is greatly appreciated especially my wife, Faridah Mustapha, my three kids, Anwar, Farah, and Diyana and including my mother-in-law who always spare time with the kids while my wife and I were at our universities.

Last and not least, the encouragement and prayers to The Almighty (day and night) from my father back home in Malaysia is touching and wonderful.

To those whose contributions I have neglected to note here from sheer failure of memory or character, please accept my apologies. I owe you all a great debt.

Alias Abdul-Rahman
1st. July, 2000

Contents

List of Figures

Page

Page

Page

Page

List of Tables

Acronyms

0-D	0-dimensional
1-D	1-dimensional
2-D	2-dimensional
3-D	3-dimensional
GIS	Geographical Information Systems
2D GIS	2-dimensional Geographical Information Systems
3D GIS	3-dimensional Geographical Information Systems
B-rep	Boundary Representation
CAD/CAM	Computer Aided Design/Computer Aided Manufacturing
CSG	Constructive Solid Geometry
DT	Distance Transformation
DTM	Digital Terrain Modelling
FDS	Formal Data Structure
ID	Identifier
MDI	Multiple Document Interface
OO	Object-Oriented/Object Orientation
OO DBMS	Object-Oriented Database Management System

OQL	Object Query Language
OWL	Object Windows Library
PC	Personal Computer
POET	Persistent and Object Extended Technology
SDI	Single Document Interface
SQL	Structured Query Language
TEN	Tetrahedral Network
TIN	Triangular Irregular Network

Chapter 1

Introduction

1.1 Introduction

Geographical Information Systems (GISs) represent a powerful tool for capturing, storing, manipulating, and analysing geographic data. This tool is being used by various geo-related professionals, for example surveyors, cartographers, photogrammetrists, civil engineers, physical planners, rural developers, geologists, etc. They use the tool for analysing, interpreting, and representing the real world and understanding the behaviour of the spatial phenomena under their respective jurisdictions. Almost all of the systems used by the geoinformation community to date are based on two-dimensional (2D) or two-and a half-dimensional (2.5D) spatial data. In other words, one may find difficulty processing and manipulating spatial data of greater dimension than 2 in the existing systems, resulting in inaccurate or at least very incomplete information. Furthermore, manipulating and representing real world objects in 2D GIS with relational databases are no longer adequate because new applications demand and increasingly deal with more complex hierarchical spatial data than supported by the relational model. It has been suggested in the literature that the abstraction of complex spatial data could be handled more effectively in an object-oriented rather than in a relational database environment (Egenhofer and Frank, 1989; and Worboys, 1995).

The limitations of the current 2D GISs, especially in geoscience, have been reported in the literature by such as Jones (1989), Raper and Kelk (1991), Rongxing Li (1994), Houlding (1994), Bonham-Carter (1996), and Wei Guo (1996). The limitations mentioned relate to data dimensionality and data structures. Single valued z-coordinate data such as a point (x, y coordinates) with the z-coordinate representing height presents no data handling difficulty in such systems, but it is inadequate for data with multiple z-values

(Bonham-Carter, 1996; and Raper and Kelk, 1991) such as ore bodies and other important three-dimensional real world entities. A major impediment to establishing 3D GISs is associated with inappropriate spatial data structures, as reported in the Jones (1989) and Rongxing Li's (1994) papers. These two authors have proposed voxel data structures for 3D data as a solution to the data structuring problem, but no real operational system was developed based on the structure. The problem has also been highlighted in the geological field as reported by Houlding (1994). True representations of spatial information, for example sub-surface 3D objects, could not be successfully achieved with 2D systems. 3D visualisation tools alone (for example Advanced Visualization System (AVS), Voxel Analyst from Intergraph and other Digital Terrain Model (DTM) packages) were not able to truly manage such data as demanded. For example Wei Guo (1996) experimented with the 3D modelling of buildings by using Molenaar's (1992) formal data structure in the relational database environment together with AutoCad as a 3D visualization tool; AutoCad was used to generate the building models. In the literature, a common suggestion has been that the existing GISs were able to handle most of the 2D spatial data, but had difficulty in handling 3D spatial data and beyond, therefore, an extension of the existing systems to at least a third-dimension (3D) is one of the solutions suggested by GIS researchers.

Another observation is that the literature cites no implemented work on three-dimensional GIS coupled with object-oriented technology; given that the weakness of conventional off-the-shelf 2D or 2.5D GISs are revealed when three-dimensional real world entities are considered, it is surprising that object-orientation and three-dimensionality have not apparently been jointly considered. Some work has focussed on the 3D issue, e.g. Fritsch and Schmidt (1995), Kraus (1995), Pilouk (1996), and Fritsch (1996a). But all of these attempts were based on the relational database environment. Therefore, this research looks at both 2D and 3D spatial data modelling and the development of a geoinformation system using an object-oriented technology to attempt to solve the 3D problem in the GIS environment.

The key foci of this research are 2D and 3D spatial data, spatial data modelling, spatial data construction, databasing, and object orientation converging to a 3D object-oriented GIS.

1.2 Background to the Research Problem

In geomatics we consider real world objects to exist in 3D, so it is desirable to have a system which is able to store, handle, manipulate, and analyse objects in the 3D environment. As mentioned in the previous section, current popular GIS software handles, manipulates, and analyses geographic data in 2D or 2.5D, thus if we use this kind of system to manipulate our 3D data full (particularly multiple Z coordinates) information about the objects cannot be achieved. Therefore 2D GIS (or 2.5D GIS) needs to be extended, i.e. to 3D GIS. Only within the last decade, has 3D GIS begun to be discussed in the GIS research community (Raper and Kelk, 1991; and Rongxing Li, 1994). It seems the development of this particular GIS approach has been relatively slow due to a lack of proper spatial data models and data structures, and a lack of a comprehensive theory of object relationships and data basing for the 3D environment (Wei Guo, 1996). As implied attempts have been made to develop 3D GIS; the attempts can be found in Rongxing Li *et al.* (1996), Pilouk (1996), and Qingquan Li and Deren Li (1996). The two Lis used an octree approach for 3D subsurface geological modelling, Pilouk used a 3D TIN approach for regular features on the terrain, and a combination of octree/tetrahedron was also proposed by Qingquan Li and Deren Li (1996). Others used Constructive Solid Geometry (CSG) and Boundary-representation (B-rep) approaches (Cambray, 1993; Cambray and Yeh, 1994; Bric, 1993; Bric et al, 1994; and Zeitouni and Cambray, 1995). All of this work was based on regularly shaped objects, which were man-made, and relational data basing. On the other hand, there appears very little published work on the modelling of 3D objects including natural objects, e.g. forests, plants, water bodies, and other natural subsurface features using the object-oriented (OO) approach. Recent research (Rongxing Li, 1994; and more recently Fritsch,

1996b) in this domain has suggested that 3D spatial data modelling, structuring and data basing with object-orientation leads to better 3D GIS. This suggestion seems mainly to arise from the complexity of 3D spatial data, as well as some positive features of object-orientation where every physical or spatial object of the real world can be defined during software development. It therefore seems imperative to investigate the practicality of a means to improve, in 3D, the representation of natural objects and to manage them in an object-oriented GIS.

1.3 Research Objectives

Based on the foregoing, the objectives of the research reported in this thesis have been to:

- Review various methods of 3D object representation.
    - classify the methods of representation,
    - develop criteria for choosing the most appropriate representation,
    - based on developed criteria, assess representations with regard to objects in the natural environment and choose the most appropriate.
- Develop the means to incorporate widely available digital data (2D data, DTM data, borehole data and 3D photogrammetric data) into the representations.
- Develop algorithms and implement the chosen representations of 2D and 3D spatial objects for geospatial databases in the object-oriented environment.
- Develop example GIS applications (e.g. query, display and terrain analysis programs).
- Test and assess the adapted approaches of representation and present them using an appropriate object-oriented graphical user interface.

1.4 Research Scope

The main intentions of this research are to carry out an investigation of 2D and 3D

spatial data representation, to develop a geoinformation system based on the most appropriate spatial data representation and to execute some GIS tasks and functions. The study includes spatial data modelling and data structuring for terrain features. The research also looks into the use of object-orientation technology for the data modelling and geospatial databasing. Specifically, it focuses on the development of a system based on triangular and tetrahedral data structures or 2D and 3D TIN (or TEN) data structures (in this thesis, the term 3D TIN and TEN are used interchangeably). The approaches are tested through a software package developed in this research, but also some aspects of object-oriented databasing utilising a commercial object-oriented database management system (OODBMS), called POET are investigated.

Furthermore, besides using simulated data, an attempt is also made to incorporate real data in very commonly available digital forms, namely:

i) 2D data + heights (i.e., 2.5D), and

ii) 3D photogrammetric digitised data.

This research does not develop a fully operational 3D GIS system; it is not an intention of this study to deal with and develop a full range of GIS operations such as data capture, data input, data manipulation, databasing, and visualization. What is intended is to find an appropriate 3D data model for natural and man made real world entities and to propose a GIS environment which accesses that model. Thus, as can be read in section 8.4 (Future Research) it is hoped that the way is now well paved for future developments towards effective 3D object-oriented GIS.

1.5 Research Approach and Methodology

This research follows a top down view of the GIS software construction. It begins by investigating the relevant 2D and 3D spatial object representations and then pursues implementation of the spatial GIS concepts, including data modelling and construction of the models. This utilises object-oriented technology.

The framework of this research follows:

- A review of 3D GIS development including systems offered by major GIS vendors.
- A review of 3D object representations presented in the literature. The representations of the objects are then classified.
- For each selected representation, an investigation of how it can represent the spatial object primitives point, line, surface, and solid features.
- A review of OO concepts and important terms such as encapsulation, inheritance, abstract data type, and polymorphism.
- An investigation of how OO concepts and the chosen 3D object representations can be fused to represent 3D spatial data.
- A development of computer programs for structuring each chosen representation. The programs will be used to establish basic data structures and also relationships between the primitives (i.e. topology) of the data sets. Some supporting computer programs, for example display programs, will be developed.
- A translation of the chosen 3D object representations into the object-oriented environment.
- Populating the databases by incorporating existing 2D data, and 3D data, e.g. digitized contours and photogrammetric datasets.
- Developing some GIS application programs by using an available C++ programming language and OO database management software.
- Developing the means to test the applicability of the populated database.
- Testing the developed computer programs by using the test area data sets and perform some fundamental GIS operations such as database retrieval (i.e. queries) and display.

1.6 Structure of the Thesis

This thesis consists of eight chapters based on the above proposed research framework.

Chapter 1, as being discussed, highlights the research problem, research objectives, scope of the research, research approach and methodology and the thesis structure. It introduces the 'what' and 'why' of the research.

Chapter 2, overviews the development of 3D GIS. This includes a discussion of the needs of such a system. This chapter is devoted to a discussion of previous work on the problem, and how this research contributes to the whole problem domain.

Chapter 3, discusses technical aspects of spatial data. Two-dimensional (2D) and three-dimensional (3D) models of spatial data representations such as vector, raster, surface-based, and volume based are discussed where 2D and 3D Triangular Irregular Network (TIN) structures are discussed in greater detail.

Chapter 4, discusses spatial data modelling and GIS. The contribution of fundamental and conceptual aspects of data modelling towards geo information system development is the focus of this chapter. Discussions on data models, modelling steps such as conceptual, logical, and physical spatial data modelling are included. Two pertinent spatial database schemas for spatial data, i.e. relational, and object-oriented are described.

Chapter 5, presents the development of an OO TIN-based data model. The chapter begins with the OO basic concepts, the use of OO for the TIN spatial data modelling. OO database development is also discussed.

Chapter 6, presents major algorithm developments for the construction of 2D and 3D TIN

data structures. The algorithms cover several tasks, such as Distance Transformation (DT), Voronoi Tessellations, and triangulation data structuring tasks. This chapter also covers the development of other tasks, particularly GIS applications.

Chapter 7, presents the implementation of the proposed approaches. All the tests of the algorithms and the computer programs developed in the research are carried out using a data set of the study area. This includes the development of the user interface, and the application of the approaches.

Chapter 8, concludes and summarises the findings of the research. Some suggestions for further improvement to the investigated issues and problems in this research are highlighted.

1.7 Commercial Tools

The following summarises the commercial software used in this work. These are used mainly for the purpose of visualising the output of the computation involved.

- *ILWIS version 2.2*, is a desk-top based raster and vector GIS/Remote Sensing software developed by ITC (The Netherlands). It is worth mentioning that now it is being marketed jointly by PCI Geomatic Inc. company. A raster image display module helps to display generated raster files developed in this work. Chapter 6 elaborates the use of this display routine.

- *AVS Release 4.1*, is a general purpose 3D visualization package developed by Advanced Visualisation Sytems Ltd (the U.K). It is a multi-platform system. It can be used to display 3D raster files. Chapter 6 elaborates the use of this display routine.

- *POET*, is an OO DBMS package which runs under Windows95 operating system.

It is used for an object database development. Chapter 5 elaborates the use of this tool.

- *Borland C++ 5.0*, is a C++ programming compiler used for the development of all software developed in this work.

1.8 Summary

This research proposes an OO approach to represent, model, construct and manipulate 2D and 3D spatial terrain objects within a geoinformation system. The following outcomes may be considered the main contribution of this research to GIS knowledge:

- Defining a suitable structure for representing 2D and 3D spatial terrain and subsurface objects in a geoinformation system.
- Developing and implementing algorithms for 2D Triangular Irregular Networks (TINs) and 3D TIN (or Tetrahedral Networks or TENs) with an extension for constrained triangulations so that terrain features such as lines (i.e. arcs or edges), and polygons can be handled in the system developed in this research.
- Designing and implementing object-oriented TIN-based spatial data models for the proposed subsystems of a geoinformation system.
- Implementing an object-oriented TIN-based spatial data model with a commercial OO DBMS package.
- Validating all the developed algorithms and other GIS tasks and applications by the development of a software package in a multiple document interface (MDI) windows environment.

Chapter 2

An Overview of 3D GIS Development

2.0 Introduction

In the previous introductory chapter, the importance and some of the problems of 3D spatial data modelling, developing an information system based on the 3D spatial data have been introduced. In this chapter, further discussions of several types of two-dimensional (2D) GIS systems which are related to the development of 3D GIS will be conducted. Some well established systems which are currently available in the market are also reviewed. Since data structures, data modelling and database management are important aspects of system development, all the discussions and the system overview will focus on these. A discussion on major GIS functions follows.

2.1 GIS Functions

Any GIS system should be able to provide information about geospatial phenomena. Principally, the following tasks represent some functions of a GIS system. The tasks or the functions of a GIS (Raper and Maguire, 1992) are: (1) capture, (2) structuring, (3) manipulation, (4) analysis, and (5) presentation, and can be summarised as follows.

- *Capture* is inputting spatial data to the system. Many different techniques and devices are available for both geometric and attribute data. The devices in frequent use for collecting spatial data can be classified as manual, semiautomatic or automatic and the output either vector or raster format. Detailed discussion on data capture is not covered here.

- *Structuring* is a crucial stage in creating a spatial database using a GIS. This is because it determines the range of functions which can be used for manipulation and analysis. Different system may have different structuring capabilities

(simple or complex topology, relational or object-oriented).

- *Manipulation*, among important manipulation operations are generalisation and transformation. Generalisation is applied for smoothing spatial data and includes line simplification, etc. Transformation includes coordinate transformation to a specified map projection and scaling, etc.

- *Analysis,* is the core of a GIS system. It involves metric and topological operations on geometric and attribute data. Primarily, analysis in GIS concerns operations on more than one set of data which generates new spatial information on the data. Terrain analysis (e.g. intervisibility), geometric computations (volume, area, etc), overlay, buffering, zoning are among typical analysis functions in GIS.

- *Presentation*, is a final task in GIS. That is to present all the generated information or results such as in the form of maps, graphs, tables, reports, etc.

Ideally, a 3D GIS should have the same functions as 2D GIS. However, such 3D systems are not available due to several impediments, as discussed in this chapter.

2.2 3D GIS

In this section, some problems and related issues in 3D GIS software development are reviewed and discussed. Firstly, what is 3D GIS? This type of system should be able to model, represent, manage, manipulate, analyse and support decisions based upon information associated with three-dimensional phenomena (Worboys, 1995). The definition of 3D GIS is very much the same as for 2D system. In GIS, 2D systems are common, widely used and able to handle most of the GIS tasks efficiently. The same kind of system may not be able to handle 3D data if more advanced 3D applications are demanded (Raper and Kelk, 1991; Rongxing Li, 1994) - such as representing the full

length, width and nature of a borehole (some examples of 3D applications areas are listed in section 2.3). 3D GIS very much needs to generate information from such 3D data. Such a system is not just a simple extension by another dimension (i.e. the 3rd dimension) on to 2D GIS. To add this third dimension into existing 2D GIS needs a thorough investigation of many aspects of GIS including a different concept of modelling, representations and aspects of data structuring. Existing GIS packages are widely used and understood for handling, storing, manipulating and analysing 2D spatial data. Their capability and performance for 2D and for 2.5D data (that is also DTM) is generally accepted by the GIS community. A GIS package which can handle and manipulate 2D data and DTM cannot be considered as a 3D GIS system because DTM data is not real 3D spatial data. The third dimension of the DTM data only provides (often after interpolation) a surface attribute to features whose coordinates consist only of planimetric data or x, y coordinates. GIS software handling real 3D spatial data is rarely found. Although the problem has been addressed (as mentioned in Chapter 1) by several researchers such as Raper and Kelk (1991), Cambray (1993), Rongxing Li (1994), Pilouk (1996), and Fritsch (1996a), some further aspects particularly spatial data modelling using OO techniques need to be investigated. This modelling issue is addressed in Chapter 4. The demand (or the need) for this kind of system is discussed in the next section.

2.3 Who needs 3D GIS

As in the popular 2D GIS for 2D spatial data, 3D GIS is for managing 3D spatial data. Raper and Kelk (1991), Rongxing Li (1994), Förstner (1995), and Bonham-Carter (1996) present some of the three dimensional application areas in GIS, including:

- ecological studies
- environmental monitoring
- geological analysis
- civil engineering
- mining exploration
- architecture
- automatic vehicle navigation
- 3D urban mapping

- landscape planning
- archeology

The above applications may produce much more useful information if they were handled in a 3D spatial system, but it appears that 3D spatial objects on the surface and subsurface demand more complex solutions (e.g. in terms of modelling, analysis, and visualization) than the existing systems can offer.

Another possible application of 3D GIS analysis is for the marine biologist. 3D survey systems, in the form of multi-beam echosounders can be used to detect and locate sea-bed surfaces, and underwater objects such as fish shoals and vegetation clusters. This detection cannot be carried out simultaneously but the resulting 3D data could be stored in a GIS. 3D overlay analysis (if it existed) would be, e.g. for a marine biologist, whether the fish shoal was above, below or within the vegetation cluster. 2D overlay analysis could not distinguish between these positions, while at the same time retaining the concept of the fish shoal or vegetation cluster as specific objects.

The next section reviews the modelling and data structure research carried out in the GIS research community towards the development of 3D GIS. This will be followed by consideration of the offerings from major representatives of the GIS commercial sector.

2.4 Recent Progress Made on 3D GIS

Some recent research efforts by the GIS community has focussed on how to develop 3D systems; data structures and data models are major aspects of GIS system development. These efforts are summarised below.

Much previous work done on 3D data modelling concentrated on the use of voxel data structures (Jones, 1989). This particular approach does not address spatial modelling aspects (that is also topological aspect of the data), it is only useful for the reconstruction

of 3D solid objects and for some basic geometric computations. Another of the problems with this data model is that it needs very large computer space and memory.

Carlson (1987) proposed a model called the simplicial complex. He used the term 0-simplex, 1-simplex, 2-simplex, and 3-simplex to denominate spatial objects of node, line, surface, and volume. His model can be extended to *n*-dimensions.

Cambray (1993) proposed CAD models for 3D objects combined with DTM as a way to create 3D GIS, that is a combination of Constructive Solid Geometry (CSG) and Boundary representation (B-rep).

Other attempts to develop 3D GIS can be found in Kraus (1995), Fritsch and Schmidt (1995), and Pilouk (1996). These attempts were based on the TIN data structure to represent 3D terrain objects but no reports exist on the any related aspects of using OO techniques for modelling and data structure.

Data modelling and structuring of 3D spatial objects in GIS has not been as successfully achieved as in CAD (Rongxing Li, 1994). Data modelling in GIS is not only concerned with the geometric and attribute aspects of the data, but also the topological relationships of the data. The topology of spatial data must be available so that the neighbours and connectedness of objects can be determined. There are a number of mathematical possibilities for the determination of the topological description of objects. Within the TIN approach developed in this research determination of the neighbouring triangles has been developed. The information gained from the generated TIN neighbours is useful for further spatial analysis and applications. Topological relationships for linear objects as represented by TIN edges can be established. One edge is represented by a start node and an end node. From this edge topology, a chain of edges or arcs could be easily established. For TIN data, another approach is the simplicial complex developed by Carlson. A TIN's node is equivalent to 0-simplex,

TIN's edge is equivalent to 1-simplex, a TIN surface (area) is equal to 2-simplex, and 3-simplex is equivalent to a 3D TIN (tetrahedron). The simplicial complex technique checks the consistency of generated TIN structures by Euler's equality formulae, see Carlson (1987) for a detailed discussion. An OO TIN approach is described in Chapter 4.

2.5 Commercial Systems and 3D GIS

There are few systems available in the market which can be categorised as a system which attempts to provide a solution for 3D representation and analysis. Four systems are chosen for detailed consideration. They were chosen because they constitute a large share of the GIS market and provide some 3D data processing functions. The systems are the 3D Analyst of ArcView (from Environmental System Research Institute or ESRI Inc.), Imagine VirtualGIS (from ERDAS Inc.), GeoMedia Terrain from Intergraph Inc. and PAMAP GIS Topographer. The following review is based on available literature and Web-based product reviews.

2.5.1 ArcView 3D Analyst

The 3D Analyst (3DA) is one of the modules available in ArcView GIS. In ArcView these modules are known as extensions. The system's extensions and the main GIS module, that is ArcView itself, is shown in Figure 2.1. ArcView is designed to provide stand alone and corporate wide (using client-server network connectivity) integration of spatial data (Maguire, 1999). The 3DA can be used to manipulate 3D data such as 3D surface generation, volume computation, draping for other raster images (such Landsat TM, SPOT, GeoSPOTV images, aerial photos or scanned maps), and other 3D surface analysis functions such as terrain intervisibility from one point to another (ESRI, 1997).

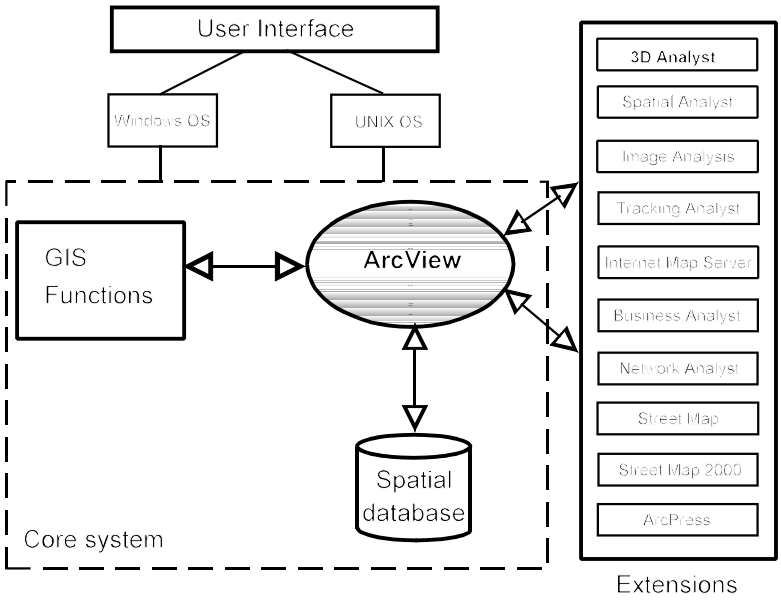


Figure 2.1 The 3D Analyst (shown on top of the extension's box) within ArcView system.

The system runs mainly on personal computers and accepts several operating system such Windows 95/98/2000 and Windows NT 4.0 as well as wide range of UNIX platforms (ESRI, 2000). The system works mainly with vector data. Although raster files can be incorporated into 3DA, but only for improving the display of vector data (e.g. by draping vector data with aerial photo images). (In this particular system, raster files are used to enhance the vector data.)

In summary 3DA can be used to manipulate 3D data especially for visualization purposes. Thus, ArcView is very much a 2D GIS system, but 3DA supplies 3D visualization and display (e.g. of data with x, y,z coordinates). 3D GIS analysis is not achieved. It is worth noting that 3DA supports the triangular irregular network (TIN) data structure.

2.5.2 Imagine VirtualGIS

It is worth mentioning that the Imagine system was originally developed for remote sensing and image processing tasks. Recently, the system has provided a module for

GIS. The Imagine system is one of the GIS solutions developed by ERDAS Inc (ERDAS, 2000). The GIS module is called VirtualGIS. It is a module that provides three-dimensional visual analysis tools. The system runs under various computer systems ranging from personal computers to workstations such as DEC computers, IBM personal computers, Hewlett Packard, Sun Sparc and IBM RISC machines. Currently the system works with operating systems such as Windows98/2000, Windows NT and various UNIX systems. It is a system which has an emphasis on dynamic visualisation and real-time display in the 3D display environment. Besides various and extensive 3-D visualizations, the system also provides fly-through capabilities (Limp, 1999). Figure 2.2 shows the system overview of the VirtualGIS with its core Imagine system.

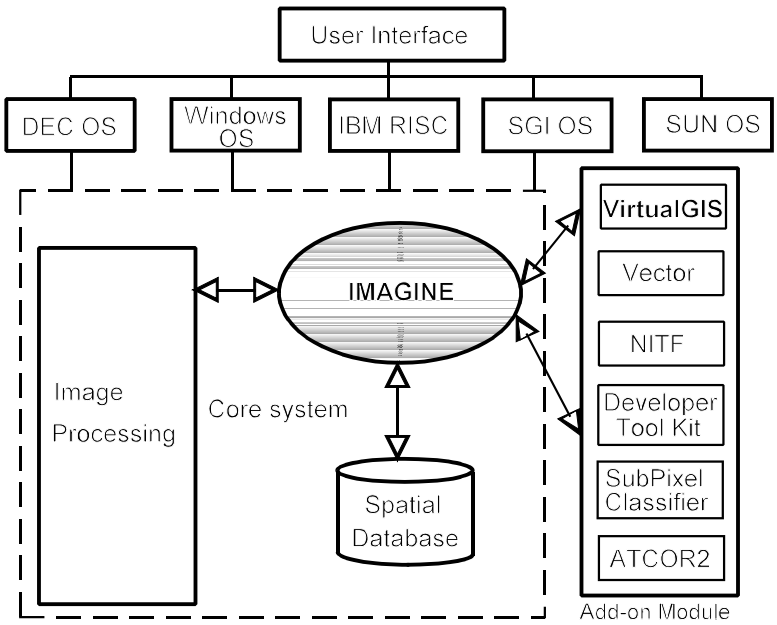


Figure 2.2 The VirtualGIS component (shown on top of the Add-on module's box) in the Imagine system architecture.

As with 3DA this system also centres around 3D visualization with true 3D GIS functions hardly available.

2.5.3 GeoMedia Terrain

GeoMedia Terrain is one of the subsystems that work under the GeoMedia GIS system

developed by Integraph Inc. The system runs under the Windows operating systems (including NT 4.0 system). The Terrain system performs three major terrain tasks, namely, terrain analysis, terrain model generations, and fly-through (Integraph, 2000). In general the Terrain serves as DTM module for the GeoMedia GIS as with the other systems mentioned in the previous sections where true 3D GIS capabilities are hardly offered by software vendors. Figure 2.3 shows the Terrain subsystem within the GeoMedia core system.

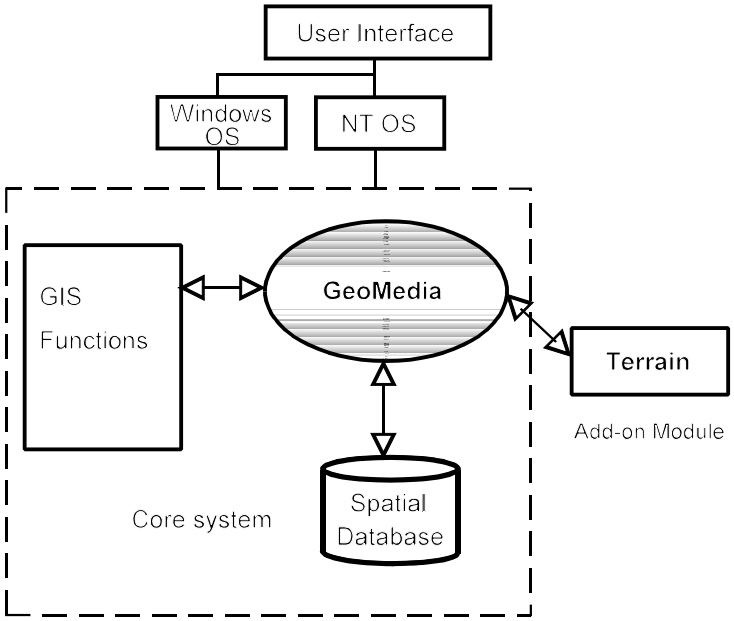


Figure 2.3 The Terrain component within the GeoMedia system.

2.5.4 PAMAP GIS Topographer

This GIS system is one of PCI Geomatics Inc.'s products. It runs under Windows95/98 and NT operating systems. PAMAP GIS is a raster and vector system (Geomatics, 2000). Besides its 2D GIS functions, the system has a module for handling 3D data, called Topographer as in Figure 2.4. Four main GIS modules are offered, they are Mapper, Modeller, Networker and Analyser which form the core system. For 2D data handling, the system performs GIS tasks as in the systems mentioned earlier. For 3D data, most of the 3D functions in the Topographer work as by any DTM packages, for example terrain surface generation, terrain surfaces analysis (e.g. calculation of area, volume) and

3D visualisation (such as perspective viewing). This system also has no solution for 3D GIS such as 3D overlay.

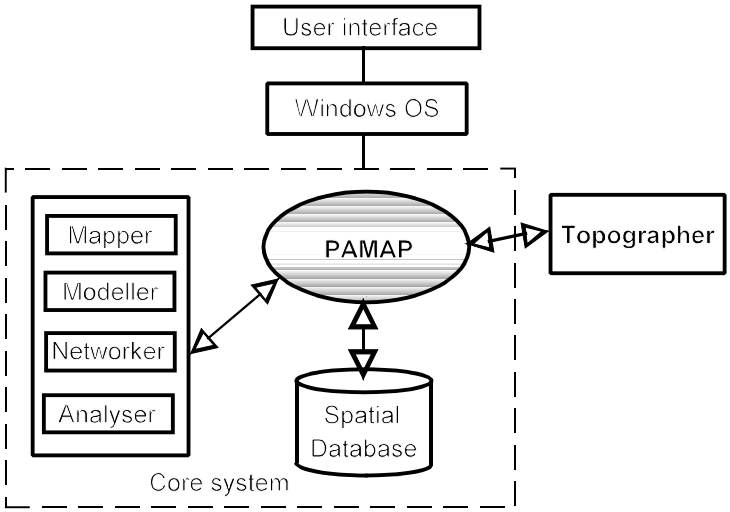


Figure 2.4 The Topographer within the PAMAP GIS system

In summary, all the systems revealed little provision of 3D GIS functionality but most of them can handle 3D data efficiently in the aspect of 3D visualization. A fully integrated 3D GIS solution has yet to be offered by any general purpose GIS vendor.

There are a few prototype 3D GIS systems and one of them is developed by the Fraunhofer Institute (Germany). This system utilises a CAD modeller which can generate 3D objects (such as buildings) on top of the terrain (Rimscha, 1997). Another prototype system which was developed by an Austrian company Grintec has tested the system within urban objects. The system, called GO-3DM also used CAD and DTM for the management of the city of Graz's 3D objects (mainly buildings) as reported by Rimscha. Despite some exciting developments in 3D visualization and the possibility of incorporating them within GIS, true 3D GIS solutions remain to be realised. This indicates that 3D GIS has far from arrived and needs further investigation.

2.6 Why is 3D GIS Difficult to Realise?

The difficulties in realising 3D GIS or 3D geo-spatial systems result from:

- Data structures: although there are several data structures available for the 2.5 D and 3D data, each of them has its own strong and weak points in representing spatial objects; and
- Data models: spatial data can be modelled in different ways. Any spatial data model should be able to describe relationships between data in such a way that topological information can be generated from them.

This thesis attempts to address these two major issues by investigating the possible uses of several data structures (including some 2D structures), the construction of these data structures, the utilisation of these structures in spatial modelling, the development of a database from the spatial data and the implementation of them in the form of a software package which can be seen as a component of GIS.

2.7 Discussion

From the foregoing discussions the problem of data structuring and data modelling for 3D data in analytical GIS environment remains unsolved. The only near solutions offered concentrate on the visualisation aspect as indicated in section 2.4. This gap in GIS functionality needs to be investigated. The effort carried out in this research work focuses on spatial data structuring and data modelling with emphasis on developing software which will contribute towards 3D GIS. To do this, several existing pertinent data structures are investigated which can handle 2D as well as 3D data. This effort is realised in the form of software development which covers aspects of data structuring, relevant algorithms' development, data modelling using object-oriented techniques and a simple front-end OO interface.

These are elaborated and implemented in the following six chapters.

Chapter 3

2D and 3D Spatial Data Representations

In the geoinformation domain, two-dimensional (2D) and three-dimensional (3D) spatial data are commonly available. There is no doubt that 2D data are utilised much more than 3D. This situation is attributable to several factors including difficulty in 3D data structuring - particularly topological data structuring (Raper, 1992; and Rongxing Li, 1994). These problems need to be investigated so that it can be seen how feasible it is to have a system capable of handling both 2D and 3D data types. This chapter focusses on the subject of spatial data representation in an attempt to contribute to an understanding of how spatial data could be utilised for a geoinformation system. The chapter's aim is to review some of the pertinent spatial data representations and adopt suitable structures for a geoinformation system capable of handling 2D and 3D spatial data.

3.1 Introduction

Geospatial data can be represented in three clearly distinct Euclidean dimensional contexts: 2D defines location by measurements on the XY axes; 2.5D defines location in 2D space with a dimensional attribute value attached to the XY location; for instance elevation above datum (Z coordinate) may act as the attribute value; 3D defines location extending through 3D space described by X, Y, and Z axes (Raper, 1992). These locations position real-world spatial objects which could be regular or irregular in shape. Man-made objects, e.g. buildings are examples of regular objects. Terrain surfaces, forests, sea floors, trees and vegetation clusters are examples of irregular objects. All of these real world objects are three-dimensional (3D). How can we retrieve elegantly these 3D objects from a system where information regarding the state, behaviour and the topological relationships of the objects with their neighbours are represented? No straightforward answer to this question exists. In GIS, spatial objects are represented in the form of points, lines, and surfaces. These primitives work well for two-

dimensional (2D) objects as described by Peucker and Chrisman (1975), but these authors did not consider 3D objects at all. As the demand from GIS applications in the 3D environment is increasing, the basic forms (e.g. single z-value for an xy location) of data representation are no longer adequate (Raper and Kelk, 1991). As a result, work has emerged attempting to solve the problem, but much has focussed on regular objects (Cambray, 1993 and Bric *et al.*, 1994) such as buildings, houses, etc.

Representing non regular objects need different data representations so that the general shape of objects can be represented. The following sections look into several existing types of representation that can be used for 2D and 3D data.

3.2 Classes of Object Representations

As an initial classification, object representations may be described as surface-based and volume-based (Rongxing Li, 1994). Li called an object a surface-based representation if the object was represented by surface primitives. It is volume-based if an object's interior is described by solid information. Figure 3.1 shows the two categories of spatial object representations.

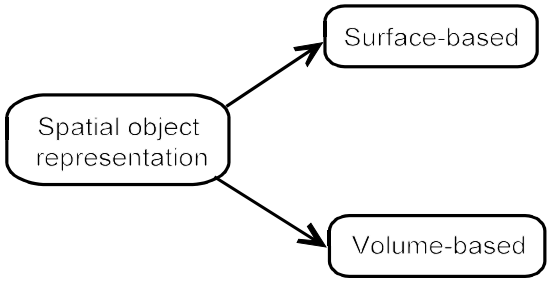


Figure 3.1 The two categories of spatial object representations.

The surface-based representations are: grid, shape model, facet model, and boundary representation (b-rep). The volume-based representations are: 3D array, octree,

constructive solid geometry (CSG) and 3D TIN (or TEN). Some of these representations are common in computer-aided design (CAD) systems but not in GIS, however they are referred to in later sections of this chapter.

The following sections describe the surface-based representations.

3.2.1 Grid

A grid is a widely used method for surface representation in GIS, digital mapping and digital terrain modelling (DTM). It is a structure that specifies height values at regular locations, see Figure 3.2. Many DTMs and terrain surface packages are based on this representation for generating surfaces as reported in Petrie and Kennie (1990). This structure has several advantages, e.g. is simple to generate, furthermore topology information (in terms of neighbour pixels) is implicitly defined (Peucker, 1978). (In this structure, the topology of grid points can be easily determined since each grid point is relative to other points). The structure may be considered as an array structure in computer programming. Each array element represents the XY locations of the grid.

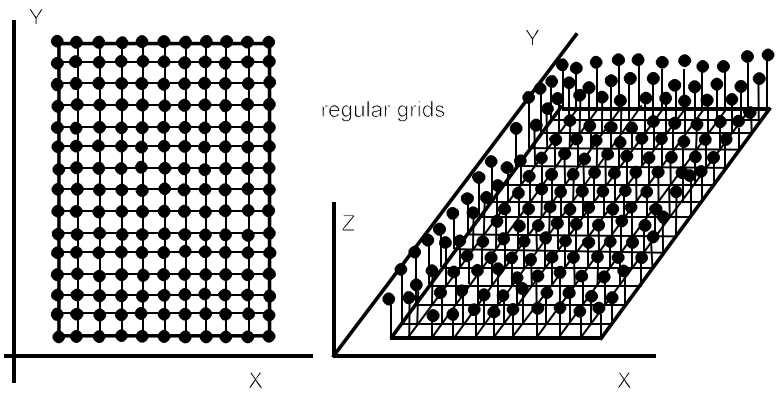


Figure 3.2 Grid representation of surfaces (orthogonal and perspective views).

The relative positions (i.e. the topology of which are the neighbouring points) of the grid points are easily defined, and it could be regular or irregular. Although excellent

terrain surfaces can be derived with this structure, it not helpful for surfaces of multiple heights, e.g. vertical walls or overhangs (Heitzinger and Pfeifer, 1996). In fact, this is one of the major drawbacks of the structure. Although it can represent surface points well, incorporating other terrain objects or terrain breaklines such as linear, polygonal, and even more complex features needs extra geometric computations and interpolations with the grid points. It is therefore the case that a better model than a grid is desirable.

3.2.2 Shape model

A shape model describes an object surface by using surface derivatives (e.g. slopes) of surface points (Rongxing Li, 1994) as shown in Figure 3.3. In this model, each grid point has slope value instead of Z value. With known slopes, a normal vector of a grid point can be defined and used to determine the shape of the surface. An experiment reported by Rongxing Li (1994) showed that the structure has an application in surface model reconstruction especially for sea bed surfaces mapping.

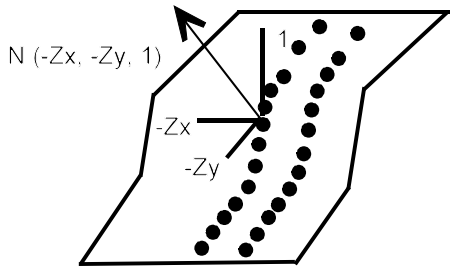


Figure 3.3 An example surface determination using shape model (after Rongxing Li, 1994)

The technique is now used for sea bed surface mapping and the usage of such a technique for land surface mapping may need to be investigated especially for data capture techniques involving very large number of points such as LIDAR. In this technique, slopes of grid points are determined by using calculation common in image processing (details can be found in Rongxing Li (1994)). This model works with regular

or irregular XY locations as with the grid approach, and thus it has the same surface mapping capability as for the grid (discussed in section 3.2.1).

3.2.3 Facet model

A facet model describes an object's surface by planar surface cells which can be of different shapes and sizes. One of the most popular facet models uses triangle facets, sometimes known as a triangular irregular network (TIN). A surface can be described by a network of triangle facets. Each facet consists of three triangle nodes which have a set of x, y, z coordinates for each node, see Figure 3.4.

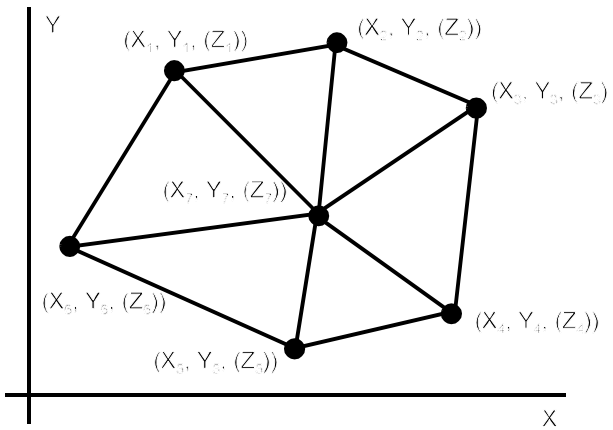


Figure 3.4 2D TIN model

Figure 3.5 shows a distribution of points on the real world. The triangle structure is widely used in DTM and other terrain surface software mainly because of its structural stability and terrain feature adaptability (Midtbø, 1996), data interpolation simplicity (Abdul-Rahman, 1992) and also for object visualization (Kraak, 1992). Triangles or TINs as illustrated in Figure 3.6 can be constructed in the raster or the vector domain, where most of the triangulations techniques are based on the Delaunay triangulations. The reader should refer to Chapter 6 (the Supporting Algorithms) for detailed discussion on the technique implemented in this work.

Briefly, one way to generate triangles in the raster domain is first by rasterising all

surface points. (These rasterised points are sometimes known as kernel points in raster data processing.) Then utilise a distance transformation (DT) technique to each kernel point. The DT calculates distances of each point to the neighbouring points. Each kernel point has its dual image, that is a Voronoi polygon of surface points. Then, from three neighbouring Voronoi polygons, a Delaunay triangle can be established (i.e. three points represent one triangle). Thus, a set of triangles can be established from a set of Voronoi polygons.

The shapes and sizes of the triangles vary, depending on the original distribution of the data sets. One of the advantages of this representation is that the original observation data are preserved, that is, all surface points are used for surface representation. An illustration in Figure 3.6 shows an example of TINs generated from random distributed points. The points were acquired using ground land survey technique (as part of a field survey data acquisition exercise carried out by students from the Topographic Science Section, University of Glasgow). It shows (in Figure 3.6) that terrain surfaces in the form of random distributed points can be represented by these planar facets (i.e. 2D TIN).

Figure 3.5 An example of terrain points (acquired by ground survey)

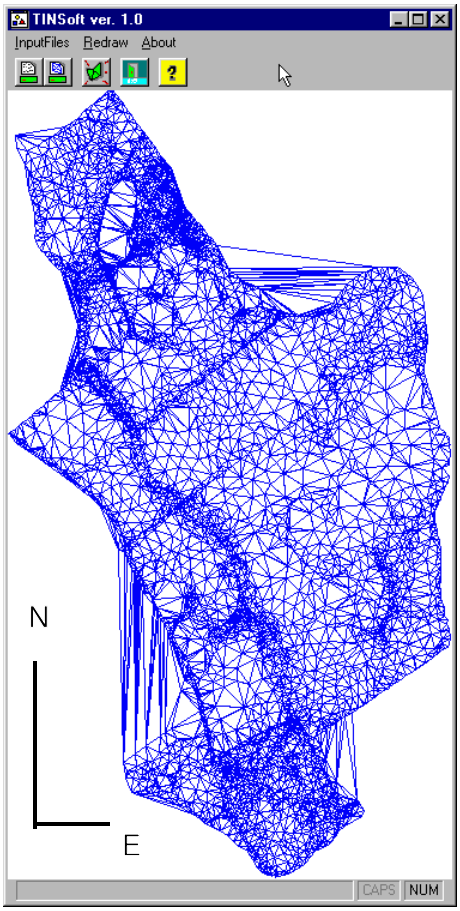


Figure 3.6 An example of TINs facet representation of terrain surfaces for points as depicted in Figure 3.5

TIN facets using digitized contours and photogrammetrically acquired data sets were also generated and are presented in Chapter 7 (the Implementation and Test chapter). The next category of surface representation is called Boundary representation (or known as B-rep).

3.2.4 Boundary Representation (B-rep)

Boundary representation (B-rep) represents an object by a combination of predefined primitives of point, edge, face, and volume. Examples of point elements are individual points, contour points and other auxiliary points which approximate a curve or a face. Examples of edges are straight lines, arcs and also circles. Examples of faces are polygon

planes and other spatial object faces such as arced faces, cone and cylinder faces. Volumes are an extension of surface elements for representing volume characteristics in B-rep. They may consist of boxes, cylinders, cones and other combinations. To represent an object by this model, an element of B-rep needs to have a geometric data element, an identification code of element and its relationship to other elements (Rongxing Li, 1994). Figure 3.7 shows a simple B-rep representation of a polygon object. Here, the key element of constructing an object is primitive combinations, i.e. a combination of points to form an edge, combination of edges to form a planar surface.

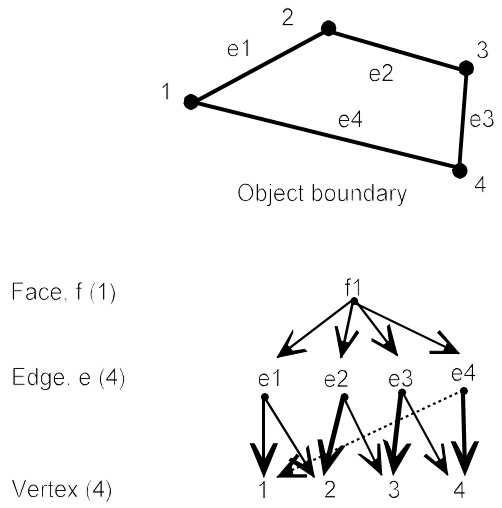


Figure 3.7 Planar polygon representation of B-rep

For non-planar surfaces, smooth surfaces functions such as a Bezier surface or B-spline functions could be incorporated in the surface generation, and this normally involves a considerable amount of geometric and complex computation. Although B-rep is popular in a computer-aided design/computer-aided manufacturing (CAD/CAM), due to computational complexity and inefficient Boolean operations, it has been suggested that B-rep is only suitable for regular and planar objects (Mäntylä, 1988; and Rongxing Li, 1994). In GIS, the use of B-rep for representing spatial objects is very limited because

the model needs to be modified in such a way that the three fundamental spatial data elements, i.e. geometric, attribute, and object identification data can be stored together with the related topological data.

The following figure (Figure 3.8) illustrates a summary of the surface-based representation of 2D objects.

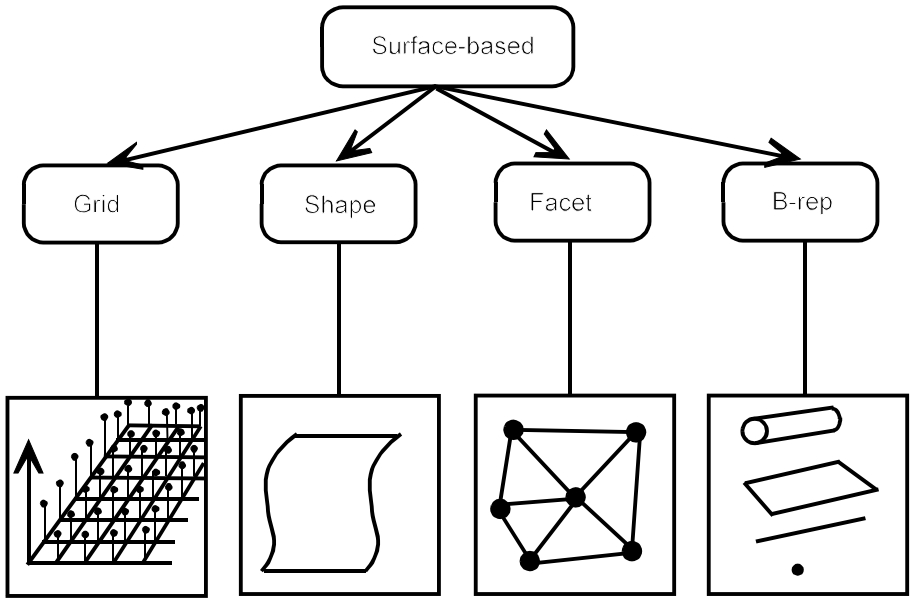


Figure 3.8 Examples of surface-based representations

The following sections describe the volume representations of 3D objects.

3.2.5 3D Array

This is perhaps the most simple data structure in the 3D domain. The structure is easy to understand and to implement, but may not be efficient for some tasks, for example if many array elements are occupied with the same values, it creates a huge but unnecessary demand for computer storage space and memory. Thus, its less suitable

for representing objects at higher resolution. Storage and memory increases with higher resolution.

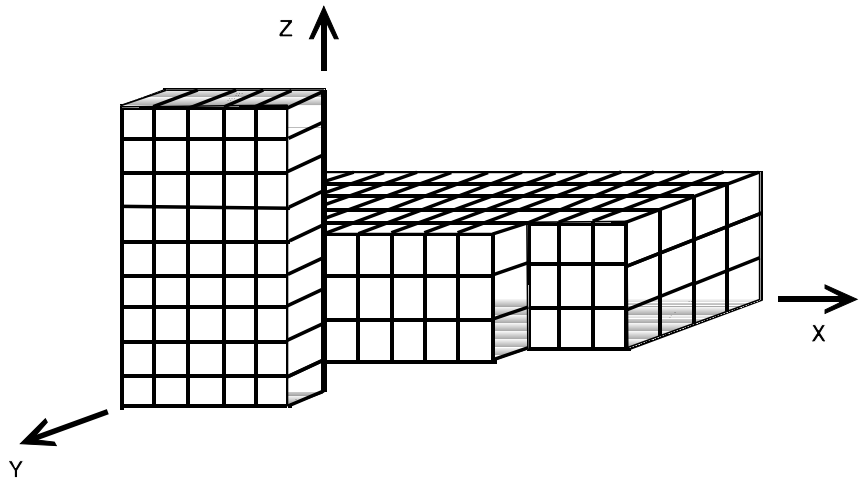


Figure 3.9 An example of 3D array representation for solid object.

In the 3D array shown in Figure 3.9, the size of the array elements is equal and each occupies the same amount of computer space although the voxel size can be specified and controlled by a program. 3D arrays need huge computing power and that is one of the reasons this kind of representation is seldom used in practice (Feng Dong, 1996).

A much better way of representing 3D objects is by varying the size of the voxel, that is the octree technique.

3.2.6 Octree

The term octree refers to a hierarchical data structure that specifies the occupancy of cubic regions of the object space. These cubic regions are often called voxels. This representation has been used extensively in image processing and computer graphics (Samet, 1984). It is a data structure that describes how the objects in a scene are distributed throughout the three-dimensional space occupied by the scene. It is simply a three-dimensional generation of a quadtree. Conceptually, the area of interest is enclosed by a cube represented by voxels (Mark and Cebrian, 1986). As in the quadtree

structure, the octree is based on recursive decomposition, and can be used to encode 3D objects (Meagher, 1982; Jones, 1989; Chen, 1991; Brunet, 1992; Rongxing Li, 1994; and Feng Dong, 1996). In the octree approach, each node is terminal or has eight descendants. The tree divides the space of the universe into cubes which are inside or outside the object. The root of the tree represents the universe, a cube with an edge of length 2^{n}. This cube is divided into eight identical cubes, called octants with an edge length of 2^{n-1}. Each octant is represented by one of the eight descendants of the root. If an octant is partially full of solid, it is termed a "grey node", and it is divided into another eight identical cubes which are represented as descendants of the octants in question. This process is repeated recursively until octants are obtained which are either totally inside the solid ("black nodes") or totally outside it ("white nodes"), see Figure 3.10. A minimum octant size (i.e. a threshold) which determines the number of subdivisions of the octants is one of the important factors in octree processing. Meagher (1982) also reported that one of the advantages of the octree approach is its simplicity for Boolean operation and visualization rendering algorithms, but it has a drawback in terms of storage space.

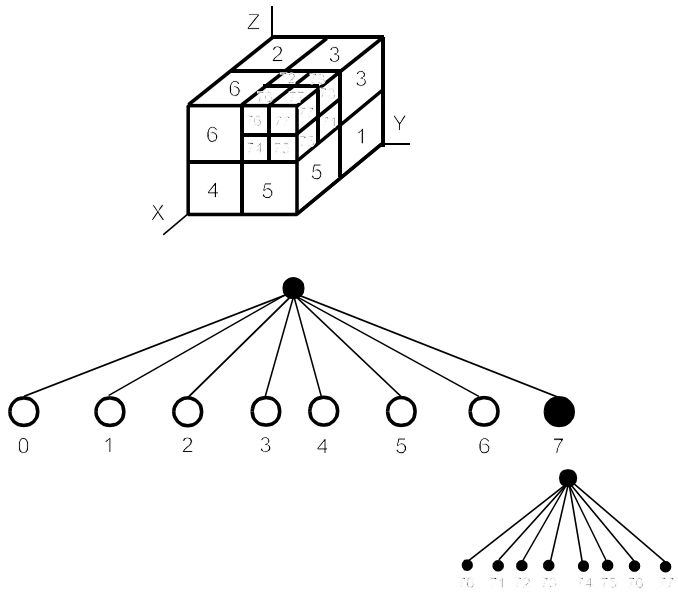


Figure 3.10 An example of octree representation of object

To represent detailed objects, a large amount of storage space and more processing power are needed. One way to overcome this problem is by using an octree model called the 'vector octree' as proposed by Samet (1984) and also reported in Jones (1989). In the vector octree, three types of octree nodes are introduced, namely, face node, edge node and vertex node. These extra nodes are used to represent object surfaces, and reduce the degree of subdivision. They, thus require less storage. Rongxing Li (1994) also reported that the octree approach is very efficient in spatial analysis, Boolean operations, and database management because of their hierarchical data structure.

3.2.7 Constructive Solid Geometry

Constructive solid geometry (CSG) represents an object by a combination of predefined simple primitives called geometric primitives, see Figure 3.11. The primitives are, for example, spheres, cubes, cylinders, cones, or rectangular solid and they are combined using Boolean set operators and linear transformations as discussed in Mäntylä (1988). CSG is commonly used in solid modelling such as CAD/CAM because object creation can be completed interactively with a simple modelling language (Raper, 1990). This representation is also widely used in engineering and architectural visualization because to construct primitives or solid geometries is usually straightforward (Feng Dong, 1996).

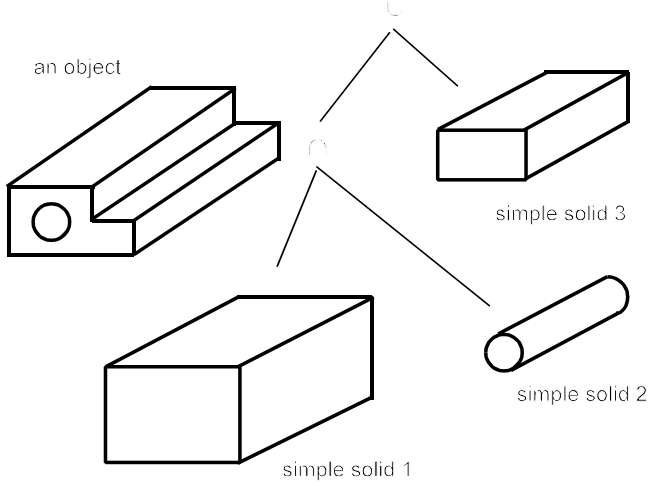


Figure 3.11 Simple object from CSG simple primitives solids

The primitives of CSG are regularly shaped volumetric instances and can be combined by using geometric transformation and Boolean operations. The geometric transformations normally involve translation, rotation and scaling and Boolean operations normally involve union, intersection and subtraction (or differencing). The storage space of CSG increases as the number of primitives increases (Samet, 1990). Previous research works suggested that CSG is only suitable for describing regularly shaped objects (Cambray 1993, and Rongxing Li 1994) because the primitive combinations of regular objects to form irregularly shaped volumetric instances needs considerable computing effort. Thus, it is therefore considered that CSG is not well suited at the moment to irregular objects.

3.2.8 3D TIN (Tetrahedral network, TEN)

Basically 3D TIN is an extension of 2D TIN, sometimes called TEN (short for a Tetrahedral Network). An object is described by connected but not overlapping tetrahedra. Similar to 2D TIN, TEN has many advantages in manipulation, display and analysis. A TEN is made of tetrahedra of four vertices, six edges, and four faces. This representation has been considered a useful data structure in earth sciences by researchers for some time (Raper and Kelk, 1991). It can be generated using the same techniques as for 2D TIN. If we build a 2D TIN from 2D Voronoi processing, then 2D Voronoi processing can be extended to 3D. 3D TIN can be constructed from 3D Voronoi polyhedrons (Qingquan Li and Deren Li, 1996) by using a 3D TIN detection algorithm, see Section 6.8. Other techniques of generating TEN can be found in Midtbø (1996).

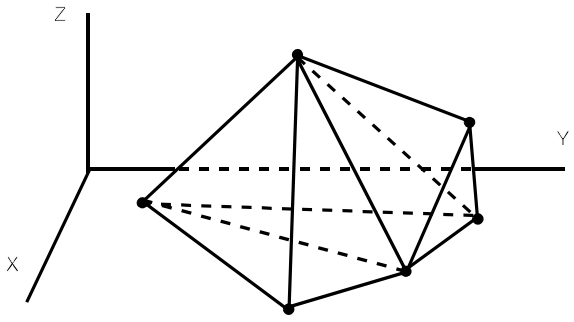


Figure 3.12 An example of 3D TIN (TEN) model

These two authors pointed out that TENs have several advantages over other solid structures. The advantages are: it is the simplest data structure that can be reduced to point, line, area and volume (solid) representations; it supports fast topological processing; and also it is convenient for rapid visualization. Work on tetrahedra for GIS is very limited. A screen shot illustration in Figure 3.14 shows an example of 3D TINs generated from simulated boreholes datasets of Figure 3.13. Each borehole has several height locations with the same XY coordinates, as the colours show.

This particular example indicates that TEN can be used to manipulate underground 3D objects such as boreholes. Volume computation of lithologies between boreholes is one of the possible 3D modelling tasks. Other applications such as iso-surface generation is also possible as demanded in Earth Science applications.

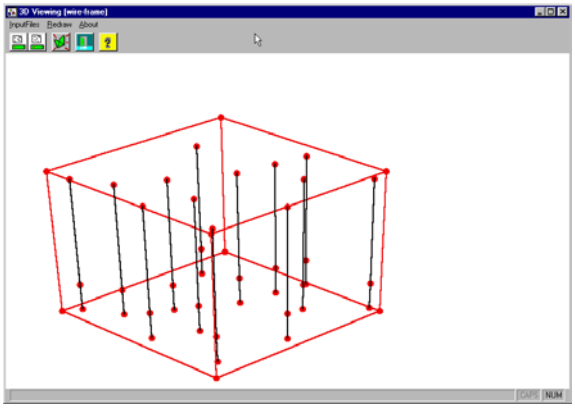


Figure 3.13 An example of simulated boreholes

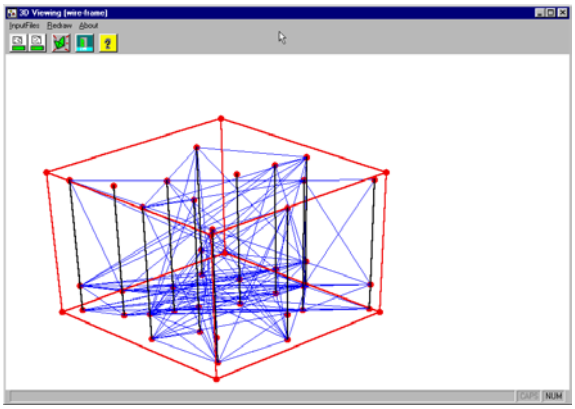


Figure 3.14 An example of 3D TIN representation for the boreholes

The visualization of 3D TIN could be made much clearer by introducing a hidden line removal module. The module has not been included in the software development.

Figure 3.15 illustrates the summary of the volume-based representations that can be used for 3D objects.

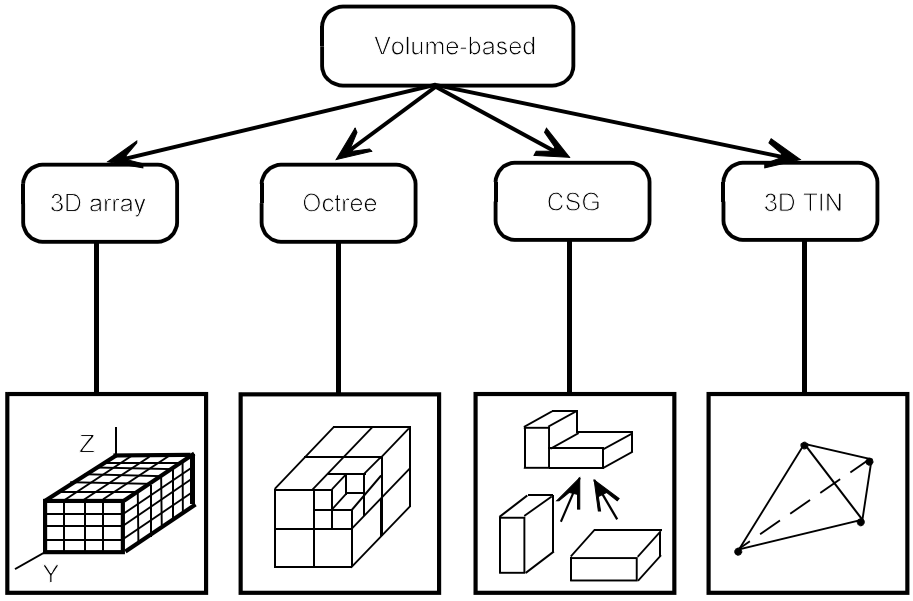


Figure 3.15 Examples of the volume-based representations

3.3 GIS Applicability of the Representations

From the foregoing discussion, it can be seen that surface-based representations describe the geometric characteristics of objects by surface entities. Grids, shape models, and facet models are suitable for describing irregular object surfaces, while the B-rep model is more for the exact surface geometry of regular shapes. For volume-based representation, 3D array, octree, and 3D TIN (or TEN) can be used for irregular objects. The 3D TIN and octree models can be used for volume objects. Compared to 3D TIN, octrees are an approximate representation and a very detailed representation of objects may be hard to achieve; furthermore with octrees, storage space increases rapidly as does the processing overhead with resolution increase. Although the storage is less of an issue these days, there is little evidence of success in using the model for spatial data representation despite the convenience in volume computation and visualization as reported by Turner (1992a).

Considering TEN (3D TIN), it is suggested that the model is able to represent objects accurately, describe complicated spatial topological relations and is able to maintain the original observations (Qingquan Li and Deren Li, 1996). Thus, we can initially assume that irregular objects can best be represented by 3D TIN and octrees. To make a choice between these two representations for irregular objects is a difficult task. The next section attempts to define some means for selecting the most appropriate representation.

3.4 The Selection Criteria

Based on the discussion and summaries of the previous sections, two representations stand out as suitable for irregular objects, they are TEN (or 3D TIN) and octree. Between the two, what is the most appropriate one? Two major items that should be considered when selecting the representation are:

- The ability to represent (or to be converted to) object primitives, e.g. points, lines, surfaces, areas and volume.

- The ability to integrate topology and attributes so that geospatial database queries, and data retrieval can be performed.

The association of these two properties within the tetrahedral (3D TIN) and octree approaches is described in the next section.

3.4.1 Representation of Object Primitives

In the real world the points, lines and areal features with which we have traditionally populated our cartographic databases do not really exist and are in fact solid (body). These representations are the simplified version of the solid objects. Some types of points and lines appear to exist as real world features but these are abstract features, such as road centre lines, boundaries, and survey control points. Furthermore surfaces as they are represented in spatial databases are a reduced description of real world

objects - being a representation of a part of the object described locationally with respect to a surface such as mean sea level or a spheroid. In reality all objects, as we perceive them and should use them at the level of detail supported by a 'typical' GIS, are irregular and

three-dimensional having more or less well defined bounding surfaces separating them from other such irregular three-dimensional objects; they are not points, lines, areas and surfaces.

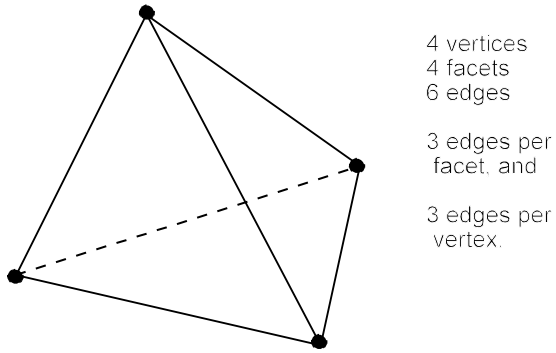


Figure 3.16 The tetrahedron (3D TIN) primitives

Given that real world objects are all irregular and three dimensional and can all be adequately represented using either the TEN or octree approaches nevertheless for reasons of efficiency or convenience the chosen data may be processed in a more primitive form (i.e. as points, lines, areas or surfaces). A GIS processing example is route selection. Thus, a consideration is needed whether either or both TEN and octree representations can be reduced to the object primitives and which representation can more easily be reduced to the required object primitives.

Figure 3.16 shows a tetrahedron, the fundamental building unit of the TEN approach. For purposes of illustration, let us consider a city, data relating to which is to be processed as if the city were a point entity. Within a city are buildings, streets and other

utilities, trees, street furniture, waterways, etc. Each of these real world objects can be represented using the TEN approach; each tetrahedron's description includes vertices and attributes. All objects belonging to the city will be appropriately attributed and retrievable, the mean of the x, y coordinates of all the vertices of these retrieved objects can provide an x, y coordinate pair to allow representation of the city as a point feature.

Considering a particular street represented by tetrahedra, for each tetrahedron at least one facet will represent the street surface, and the vertices of this facet will be points along the street. A centre-lining procedure can generate a line from the set of vertices from all the surface facets of the street's tetrahedra.

Considering as an object a piece of undeveloped land within a city represented using the TEN approach, some tetrahedron edges will be the edges between two 'undeveloped land' tetrahedra, and some will be the edges between undeveloped and land of another category. Those edges representing change in categories are the edges of the undeveloped land, and the x, y coordinates of their vertices represent the bounding polygon (or 'area') of the undeveloped land.

Finally considering the surface of the city itself, if this has been described by a series of tetrahedra, as with the street some facets will be surface facets and their vertices describe an irregular DTM. It is possible that the coordinate system used to describe vertices' locations is not appropriate for the DTM (e.g. with respect to an inappropriate datum). An appropriate coordinate transformation will need to be introduced.

Octree works with 3D raster data sets. It is therefore the case that all object entities have to be converted into 3D raster for further processing. These objects then need to be decomposed into point, line, surface, and solid primitives if they are to be used in a GIS, for example. A number of authors have reported on the use of octree for GIS, but most of them have focussed on visualization and volume computation tasks (Chen, 1991;

Mark and Cebrian, 1986; Meagher, 1982). Work on octrees with the related aspect of spatial data modelling is little reported. Much research work on octrees was for solid modelling and visualization purposes as reported in Turner (1992b).

From the foregoing discussion it can be seen that because little successful work has been done on representing primitives in the octree environment and the potential for data degradation arising from the need to interpolate the raw data to generate the octree structure, TEN representation provides a more promising model for 3D spatial object representation than octree.

3.4.2 Topology of spatial objects: simplexes and complexes

In GIS, besides geometric and attribute data, topology has a vital role in spatial information. Topology is used to determine the connection relationships of objects in space. For example, in the case of a point object, one may need to know its relationship with neighbouring objects (where it could be with points, lines, areas, or solid objects). The same holds for lines, areas, and solid objects. A number of researchers have looked into this topological problem, including Frank and Kuhn (1986) and Worboys (1995). These particular researchers use the terms complex and simplex for describing the topological relationships of planar objects. In the 2D case, triangular irregular network structures can be regarded as simplicial complexes in a Euclidean plane. Here, a 0-simplex is the set of a single point in the Euclidean plane. A 1-simplex is a straight line segment. A 2-simplex is a set of all the points on the boundary and in the interior of a triangle whose vertices are not collinear. These simplices are well represented in the facet model of representation (see section 3.2.3) where a TIN node is topologically equivalent to 0-simplex, the edge of a TIN is topologically equivalent to 1-simplex, and a TIN area (surface) is topologically equivalent to 2-simplex. Since this simplicial complex theory is extendable to n-dimension, then we could also represent TEN primitives using the same principle. That is a 3-simplex is a volume which is a tetrahedron; see Figure 3.17 for an illustration.

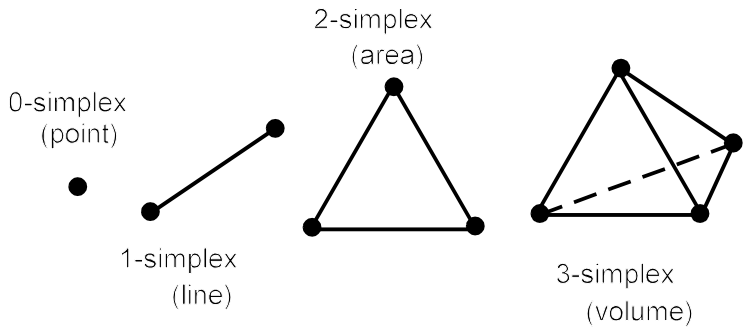


Figure 3.17 Example of simplices (0, 1, 2, and 3 simplex)

Simplices are the building blocks of a larger structures, the simplicial complexes. Complexes are built from simplices. If we recall the TIN representation (see Figure 3.4), a simplicial complex can be formed (i.e. two-dimensional complexes). This concept of simplicial complex provides a sound framework for analysis of the topology of a mixture of points and edges in a plane and is workable for the TIN representation of spatial objects (both 2D and 3D) as cited in Worboys (1995).

3.5 Vector and Raster Representation

Geoinformation data may come in vector form, raster form, or in both forms. Spatial objects are said to be in vector form if they are represented by one of the basic discrete entities such as points, lines, and areas (polygons) which are spatially referenced by a Cartesian coordinate system (Burrough and McDonnell, 1998). The same spatial object entities can be represented in raster form if they can be decomposed into pixels. Each pixel is referenced by row and column positions. Representing spatial objects as raster or vector has advantages and disadvantages. Vector representation easily offers better representation over raster because entities are represented by exact coordinates in space and do not have their locations generalized to a pixel. Thus, raster may give more approximate locations for the represented entities. But one must bear in mind that this is not always the case, it depends on several factors e.g. data collection techniques, resolution. Further comparisons of these two representations, such as based on handling

topology, reveal topology is explicitly described in the vector form and therefore this form is good for tasks such as network analysis. Geometric data processing such as coordinate transformation is difficult in raster (requiring resampling) but easy to perform in vector form (Burrough and McDonnell, 1998). A further debate on these two representations can also be found in (Antenucci *et al* (1991) and Chou (1996)).

The choice between the two representations depends on factors such as processing speed, level of difficulty, etc. In this research we used the raster form as a means of data processing for 2D and 3D TIN model construction and also for the related data structuring. That is due to the simplicity of raster data processing. The discussion in section 3.2.3 indicated that TINs could be constructed using rasterised datasets. The simplicity of raster data processing for the two object representations is also examined in Chapter 6 (the Supporting Algorithms chapter).

3.6 Summary

From the foregoing discussion of 2D object representations, 2D TIN has been shown to have several advantages over the other models of the same category (i.e. the grid, shape, and the B-rep.). The model's promise relies on the fact that it can be used to construct a generic data structure (including topological relationships). Other models such as grid, shape, and B-rep require further structure modifications before they can be utilised, and thus they lead to expensive modelling in the digital environment.

Since 2D TIN can be extended to 3D TIN and have similar geometric properties, 3D TIN can represent 3D spatial objects. An important property of the model (or the structure) is that simple object primitives are aggregatable into a larger object. The aggregation of features into more complex features is perhaps the most important feature in spatial data modelling. Models other than 3D TIN have some drawbacks in this task, e.g. they require huge computing effort. For example, real world spatial objects are complex in nature and it is obvious that tremendous decomposition operations are involved if one

dealt with them as octrees. Although the octree approach is widely used in the solid geometry visualization community, difficulty in spatial data structuring and the related topology entails limited practicality in GIS.

The pertinent spatial object representations have been described and TINs (2D and 3D) have been identified as the most appropriate representations for the 2D and 3D spatial objects. Thus, these structures become the major focus for the development of a geo information system in this research.

The modelling and other relevant aspects of the geoinformation system are discussed in the next chapter.

Chapter 4

Fundamental Aspects of Spatial Data Modelling and GIS

4.1 Introduction

In general, a GIS can be considered to have several components such as spatial, graphical, numerical, and textual components (Worboys, 1995). These system components have several important building blocks such as data modelling, data structures, and types of applications. However Molenaar (1996a) reported that it is the process of spatial data modelling alone which leads to the development of a complete geoinformation system. This chapter introduces the fundamental concepts of spatial data modelling and GIS. As well as the concept of modelling spatial data being investigated, the construction, manipulation and management of spatial data within the development of a GIS system are also investigated; in particular the concepts of spatial data, modelling of spatial data, construction, manipulation and management of spatial data in the domain of the triangular irregular networks (TINs) data structure are foci of this chapter. The aim is to describe major processes and steps involved in the development of a system which is based on TIN spatial data. Although this system is far from complete (since it does not contain, for example a temporal aspect), most of the major components and the related building blocks for the system are considered. Relevant temporal aspect of GIS are addressed in Langran (1992) and Wachowicz (1999).

The layout of the proposed TIN-based system is presented at the end of this chapter, following the discussions on spatial data, spatial data modelling, data structuring, database models and the related database management systems (DBMS).

4.2 Spatial Data

Figure 4.1 shows the basic components of spatial data. Principally, there are three spatial data components that need to be stored for GIS data, they are geometric data, thematic data and a link identification (ID) for the geometric and the thematic component. The illustration in Figure 4.1 shows the link between the geometric component (it deals with the location of the data by means, for example, of a reference coordinate system) and the thematic component (it provides the attribute values of the data, e.g. names, and other identifiers (IDs) of the data). Object or feature needs to be geometrically and thematically described (Longley *et al*, 1999; Laurini and Thompson, 1991). The basic components of spatial data (TINs) can be used to describe real world terrain objects, whether natural or man-made; thus we have TIN-based spatial objects.

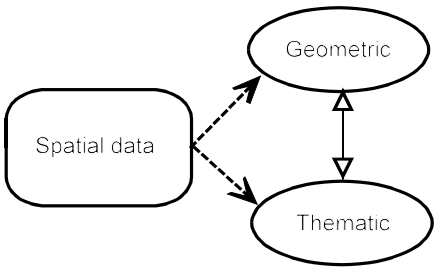


Figure 4.1 The spatial data components

4.3 Spatial Data Modelling

Spatial data modelling is a process of describing real world spatial objects so that these objects as perceived by us can be represented in a form or notation we understand and use. There are several techniques for perceiving the real world (Burrough and Frank, 1995). These techniques have different descriptive models for different levels of complexity of perception of the real world. If we would like to have these models represented and operational in a geo-information system, then they have to be mapped into data and processing models that can be handled by computers. Figure 4.2 illustrates a general view of three stages of spatial data modelling that one may apply

in information system development.

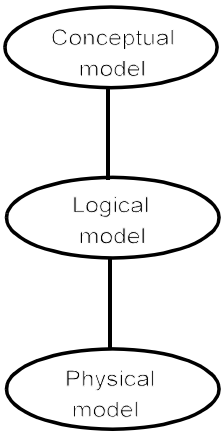


Figure 4.2 A typical spatial data modelling steps.

A data model is a notation for describing data. It is a meta concept defining the content, structure, and meaning of data. The model also provides concepts to describe the structure and contents, for example, of a database, and the goal is similar to that of the data types (either basic data types or the Abstract Data Type (ADT)) used in programming languages to describe data within programs. Data models can be classified into the conceptual data model (or high level model), the logical data model (or implementation model), and the physical data model (or low-level model) as shown in Figure 4.2. Conceptual data models provide easy to perceive high-level concepts. They are used in the early stages of system development to communicate between end-users and system designer. Physical data models provide low-level concepts to describe how data are stored and accessed in the computer. The logical data model bridges the gap between the conceptual data model and the physical data model. It is sometime known as an implementation data model. It is used by database management system to implement reality in computerised databases. The illustration in Figure 4.2 shows the steps in typical database design and also serves as a basic means to model terrain spatial

objects.

In the case of terrain spatial objects, they need to be divided into several classes. For example, we may classify them as point features, linear features, area features or body or solid features. Each of these classes of object has geometric and thematic components with their corresponding object identifiers (IDs). Terrain objects could be represented by several techniques as discussed in Chapter 3 where it was shown that TIN data have some promising structures and provide a fundamental framework for modelling spatial objects in this study; see Chapter 6 (the Supporting Algorithms) for the construction of the structures.

The following section describes the conceptual modelling of spatial objects with TIN data structures. The structure has been widely utilised for terrain surfaces modelling in popular DTM and GIS software. Figure 4.3 shows a simple relationship between real world objects (spatial objects) and the TIN primitives (i.e. node, edge, surface, and body (3D TIN)). The spatial objects could be divided into four basic entities. They are point entities, line entities, area entities, and solid entities.

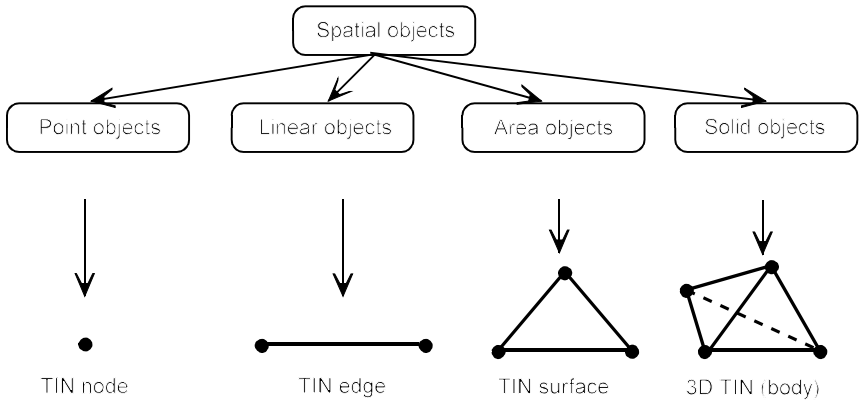


Figure 4.3 TINs representations for spatial objects

Figure 4.3 shows the relationships between the four classes of objects and the TINs primitives of node, edge, triangle, and tetrahedron. Here, the term "object" represents

the real world object, not the "object" that is used to describe an instance of a class in an OO programming language. The word "entity" is frequently used for this type of "object". Principally, a TIN triangle has 3 nodes and 3 edges (or lines). The figure also shows that point objects are represented by TIN's nodes. Linear objects are represented by TIN edges, area objects represented by the triangles, and solid objects are represented by 3D TINs (i.e. tetrahedra). These four classes of object need to be put in a model so that the relationships between them can be established. Molenaar (1989) introduced such a spatial model, shown in Figure 4.4. This model does not address 3D space.

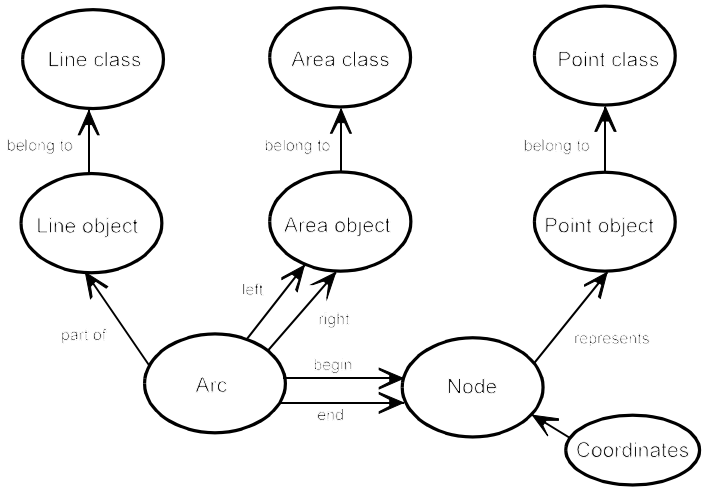


Figure 4.4 A spatial model (after Molenaar, 1991)

This model consists of:

- three object types, namely Point object, Line object and Area object are classified according to the geometric description of spatial objects; and
- three geometric data types (or geometric primitives). They are node, arc, and also including the coordinate. The "Coordinates" is only used to store all the coordinated points.

The model represents point objects using nodes and line objects using arcs. Area objects

are represented with chains of arcs. The model also assumes that an arc is represented by a straight line of two end nodes (i.e. begin and end nodes). The same assumption is made in this work, that is an edge is represented by two end points, and a polygon is formed by a series of edges. In the above model, the relationship between geometric and thematic aspects of objects is indicated by arrows. That is to say the arrows between (the "Arc", the "Node", and the "Coordinates") ellipses and (the "Line object", the "Area object", and the "Point object") ellipses. That is the links between the second and the third row of the ellipses. By using the above model, further links and rules controlling the relationship between these features could be established.

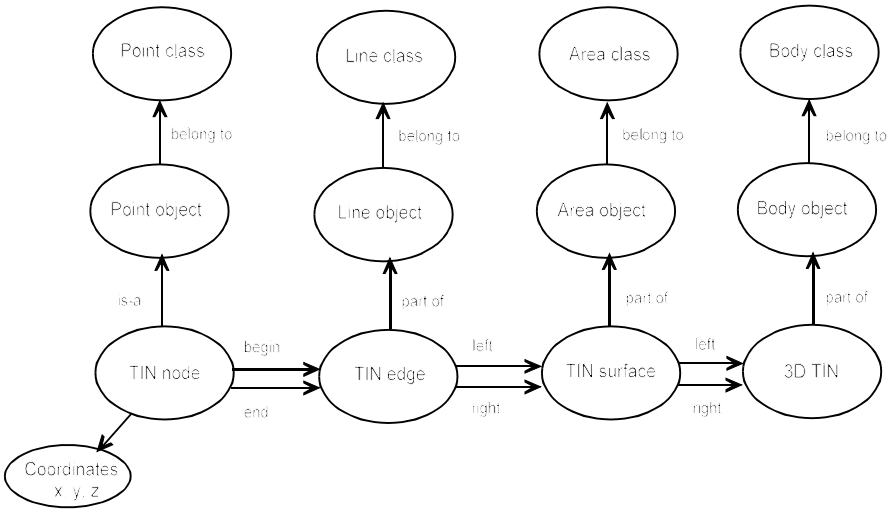


Figure 4.5 The TIN-based spatial data model (after Fritsch, 1996a)

To handle 3D objects the model attributed to Molenaar (1996) and Fritsch (1996a) is adapted, shown in Figure 4.5. This model is an extension of the model illustrated in Figure 4.4. In this 3D formalisation model, 3D objects are represented by 3D TIN primitives (i.e. tetrahedra). TIN nodes represent point objects, TIN edges represent area objects, TIN surfaces (triangles) represent area objects and 3D TINs represent solid objects. In the research reported in this thesis, the 3D TIN primitive is used, for example to represent a region bounded by a series of boreholes, as indicated in Chapter 3 (section 3.2.8). The topology of the primitives in the model is clearly shown, that is by the

indication of "begin"and "end"arrows of TIN node to TIN edge ellipses. The "left" and "right" arrows of TIN edge to TIN surface (area) ellipses. The same applies to the 3D TIN, it involves left and right topology indicators from the previous TIN surface ellipse. The topology of the primitives (e.g. left or right) in relation to other primitives has been implemented as described in Chapter 6 (section 6.4.1, section 6.10.1). The model does not support the "inside" topology of the TEN. By "inside" topology of the TEN is meant relationships of the features or objects inside of the TEN e.g. fault lines through an ore body. It can handle the "outside" topology of the TEN by a combination of several structures implemented in Chapter 6.

The adapted data model has to be translated into a workable data structure by the DDL (dynamic data linking) of the DBMS as discussed in Chapter 5.

4.4 Spatial Data Structuring

Spatial data structuring has a crucial role in spatial modelling. It organizes spatial data into a form suitable for computers. It sometimes can be regarded as being intermediate between the data model and a file format. Any adopted data structure eventually leads to the provision of relationships and linkages between data components in a system such that useful spatial information on objects can be generated. Spatial data structuring is also known as physical modelling where the adopted model is implemented in the form of computer programs. In the foregoing the two most common data structures used in GIS, raster and vector, have been briefly mentioned. Different structures are used for different tasks, depending which are the most efficient and suitable. In this section the data structures used for the software development in this work and which are also in the form of raster and vector data structures are described. In this work, for visualization purposes, two commercial GIS packages file formats ILWIS and Arc/Info and the AVS package for 3D visualization were adopted.

Figure 4.6 shows the two geometric structures of spatial objects, namely the raster and

the vector structure and these are discussed in the next section.

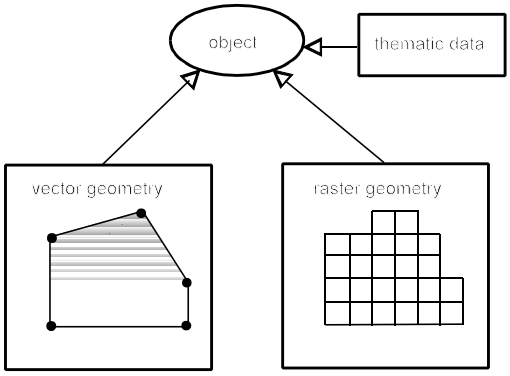


Figure 4.6 Two geometric structures of spatial objects (after Molenaar (1996b))

4.4.1 Raster structure

The raster data structure is sometimes known as the row and column data structure. Spatial objects are divided into rows and columns of some specified size depending on the resolution of the original datasets or the required smallest pixel size. In this work a raster data structure has been used only for rasterization, distance transformation, and Voronoi tessellations. Although, in principle the structure can be used to generate a spatial information system, it has not been utilised further in this work. The raster structure shown in Figure 4.7 is only used for constrained triangulation purposes (detailed discussion is provided in Chapter 6, section 6.9.2).

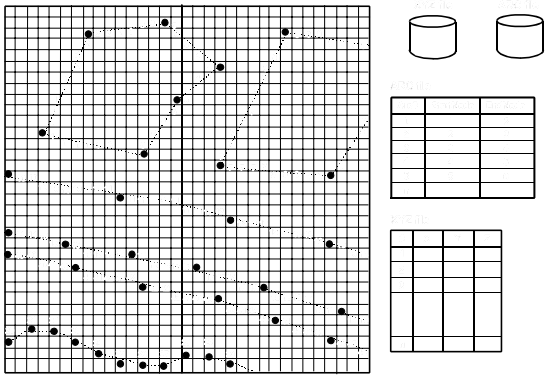


Figure 4.7 An example of the pixel locations of the rasterized points and several edges or arcs. Tables on the right represent the coordinates and the arc files.

The vector data structure is another type of structure that can be used for developing a spatial information system. There are several types of vector structure, they can be categorised into the spaghetti model, the topological model and the TIN-based (2D and 3D) type. They are considered in the next section.

4.4.2 Vector structure

Spaghetti model

The spaghetti model represents a simple object configuration, for example a polygon is represented by a series of straight lines and points. There is no explicit representation of the topological relationships of the configuration, such as adjacency relationships between lines and points to form polygons or other neighbouring objects. The model provides basic connectivity of objects such as a line is formed by two nodes. Some forms of geometric computation may be required in order to obtain more advanced object relationships. In this work, a spaghetti model has been used to store point and line objects. A simple connectivity of points form an arc and connections of arcs form a polygon. Although the model is less efficient in terms of storage space because of duplication of points stored more than once around a polygon, it served its purpose in this work. The model is also useful for simple geometric operations where topological

relationships are not required. Other mapping tasks that may benefit from such a model are, for example, automated cartography and digital mapping when complex spatial relationship searching is not required.

A more useful model than the spaghetti model (the topological model) is described in the next section.

Topological model

The topological model is considered a much better model over the spaghetti model for several reasons. This model is able to handle topological relationships between spatial objects; for example - what is on the left and what is on the right of an object? This particular topological relationship between objects gives further useful information about an object. Most of the operational GIS packages utilise this model. Although different topological models may be implemented in different GIS systems, each model should be able to establish the relationship between objects in the spatial domain for the generation of connectivity information. Without topology, a system contains limited spatial information. In this work, the topological relationships between object primitives are generated by the creation of several structures. These structures give the geometric information on the objects. Attribute information on the objects are stored in different files. The attribute information can be linked to their geometric parts by OO encapsulation techniques as discussed in Chapter 5.

The data structure for the TIN model is described below. The structure can be considered as one of topological vector structures.

2D TIN model

The 2D TIN model is another type of topological model. Different TIN models can be generated, Figure 4.8 is one of the possibilities for structuring such spatial data. A program for establishing TIN neighbour information which is part of the topology has also been developed (section 6.4.1 of Chapter 6). Neighbouring triangles can be

determined for any given triangle. This topological information is useful for TIN-based data structure applications.

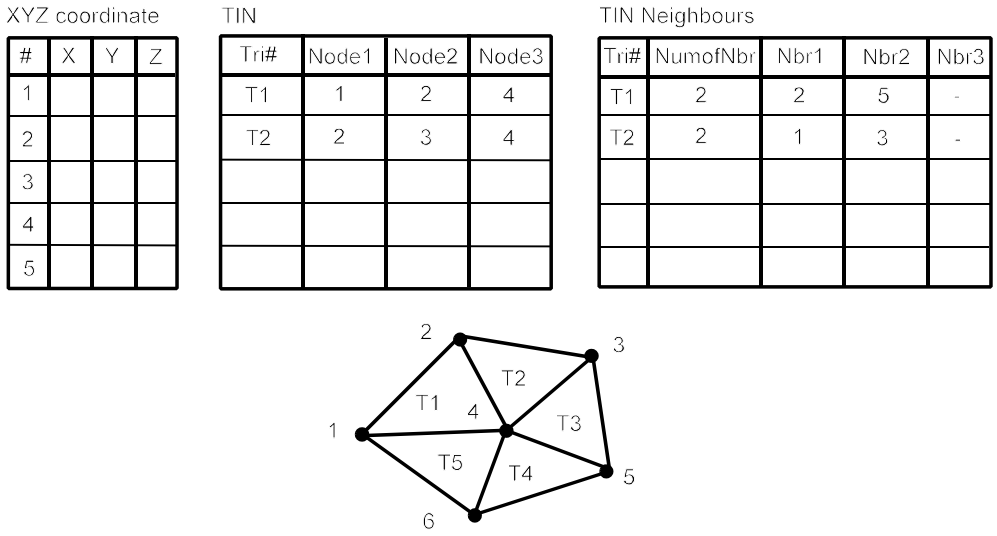


Figure 4.8 TIN model with the related topological data (i.e. tables of XYZ, TIN, and TIN neighbours)

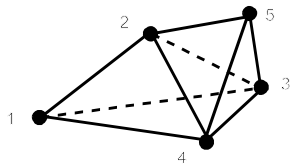
In the above diagram, a triangle, say T1 has three nodes (i.e. Node1, Node2, and Node3). The Node1 represented by 1, Node2 represented by 2, and Node3 represented by node 4 (in T1). This triangle T1 has two neighbouring triangles (i.e. T2 and T5). Their corresponding nodes for these triangles are clearly shown in the diagram. The third table (called TIN Neighbours) shown on the right gives details about the neighbouring triangles (indicated by Nbr1, Nbr2, Nbr3) and how many neighbours a particular triangle has (i.e. NumofNbr), as indicated in the diagram. Detailed explanation of these tables is provided in Chapter 6 (the Supporting Algorithms chapter). However it can be stated that the model has the following tables: a list of xyz coordinates, a list of a triangle's three nodes and a list of a triangle's (maximum) three neighbours. An arc's table is also created. It is a list of arc# and two nodes of an arc (a start node and an end node for each arc). The arc's table enables linear features to be incorporated into the modelling of spatial objects. The functionality of the TIN model developed in this work is demonstrated by translating it to a spatial database schema and implementing GIS and terrain surface analysis applications. Applications related to terrain surface analysis

include such as such as contouring, slope and aspect of surfaces, hidden-line removal in perspective plots and surface shading. Applications which are related to GIS include the area computation of a region covered by several TINs, object retrieval from a database and display.

There are several ways of storing TIN topology, Figure 4.8 shows one of the possibilities. Here, a triangle is the basic spatial object and each has three nodes. These nodes have numbers corresponding to nodes number (i.e. node#) of the coordinated points in a separate file. A topological link to the neighbouring (adjacent) triangles completes the description of the TIN structure. An algorithm to generate this topological information is discussed in Chapter 6.

3D TIN (or TEN) model

The 3D TIN model is an extension of the 2D TIN model. Similar techniques could be used to construct the 3D model as the 2D could. 3D spatial objects could be represented by 3D TIN as it has been described in section 3.2.7. In this model, a tetrahedron where each has four triangles is the basic spatial object. The model has its topology based on a list of four nodes for each tetrahedron. A list of three nodes for each of the four triangles of the TEN, and a list of coordinated points for the corresponding nodes provides the geometry of the model as illustrated in Figure 4.9.



TEN table

#	Nd1	Nd2	Nd3	Nd4
1	1	2	3	4
2	2	3	4	5

TIN table

#	Nd1	Nd2	Nd3
1	1	2	3
2	2	3	4
3	1	2	4
4	1	3	4

Figure 4.9 The TEN model with two adjacent TENs

The modelling 3D

spatial objects such as ore bodies from borehole data, in a way useful for data manipulation in the Earth Sciences community could be performed as illustrated in Chapter 3 (section 3.2.8). Surface (e.g. underground surfaces) generation is also possible by using this model.

The next section introduces two pertinent database schema for spatial objects. A database is used to store spatial objects which have been modelled by the techniques as discussed in the previous sections.

4.5 Relational Database Model

The relational model introduced by Codd in the late 1960's has been implemented by many commercial database management systems such as DBase, EMPRESS, INFORMIX, INGRES, ORACLE, POSTGRES, SYBASE, SYSTEM R, and UNIFY (Schrefl and Bichler, 1995). It represents data in a database as a collection of relations. A relation can be thought of as a table of values representing a set of similar real world objects and their relationships. The rows of a table, called tuples, define real world objects or relationships between real world objects. The columns of a table represent attributes and contain attribute values. The ability to define operations on relations whose results are again relations and support for powerful declarative languages such as SQL (Structured Query Language) are the main reasons for the popularity of the relational model. Although this model is well accepted in the non-spatial community, it has drawbacks for the handling of large and complex spatial data sets in GIS. In geoinformation practice, the pure relational geospatial model has not up to now been widely adopted because of its unacceptable performance limitations (Healey, 1991). Problems arise because of: (1) slow retrieval due to multiple joins required of spatial data in relations; (2) inappropriate indexes and access methods which are provided primarily for 1-dimensional data types by general-purpose relational systems; and (3) lack of expressive power in SQL for spatial queries (Worboys, 1999). The first problem (slow retrieval) arises due to the complexity of spatial data as mentioned before, that is

for example, if we take a polygon - it is made up of chains of edges, and the edges are made of series of points. It is thought that this problem is much better handled in an object-oriented model (or also in an object-relational model). The second problem, related to indexes of different types of spatial data, are much better handled in the extended relational model. The third problem, the limitations of SQL, has been apparent for some time in a number of fields such as CAD/CAM, GIS, multi media databases, and other non-spatial databases (Worboys, 1999).

The relational model approach for spatial data has been thoroughly researched by Pilouk (1996). This previous work makes use of the relational model for modelling TIN-based spatial data. The work was based on several relational tables to describe spatial data relationships between data types in the domain. Eight relational tables were constructed for the modelling purposes. Although the approach works for such TIN data, the modelling is thought to be better handled using the OO approach. One of the reasons was due to the problem (1) as mentioned in the previous section, that is the problem of the multiple joins required for spatial data relations.

Recent developments in computing especially in object-oriented technology have begun to influence the way spatial data is organised in GIS. These are discussed it in the next section.

4.6 Object-Oriented Database Model

The drawbacks of the relational database management system in GIS have encouraged consideration of a new solution: an object-oriented GIS. The emergence of OO databases has been stimulated by problems of redundancy and sequential search in the relational structure. In GIS their use has been stimulated by the need to handle complex spatial entities. GIS demands more intelligent spatial data handling than that required by simple point, line, and polygon primitives. The problems of database modifications especially when analytical operations such as polygon overlay are carried out (Burrough

and McDonnell, 1998) need addressing. The object-oriented approach is applied increasingly in a number of fields, although what exactly this means can be difficult to answer as there are many different definitions, formalisms and models amongst the computing community, as been reported in Worboys (1995). In fact, the term 'object-oriented' seems quite nebulous. In the relational structure, each entity is defined in terms of its data records and the logical relations that can be elucidated between the attributes and their values, whereas in object-oriented databases, data are defined in terms of a series of unique objects which are organized into groups of similar phenomena (known as object classes) according to any natural structuring. Relationships between different objects and different classes are established through explicit links. Once the data have been encapsulated in the database the way to change them or to query them is to send a request, known as a message, to carry out one of its operations (Burrough and McDonnell, 1998).

Data used in object-oriented databases need to be clearly definable as belonging to unique entities. Given that, these databases (as with their network and hierarchical counterparts) provide very efficient structures for organizing hierarchical, interrelated data. Establishing the database is obviously time-consuming as the objects must be defined explicitly and their various links need to be established. Once this is finished, the database provides a very efficient structure for querying, especially with reference to specific objects (Worboys, 1995).

In this study, an attempt is made to utilise object-oriented database functions provided in a commercial OO DBMS for TIN spatial data handling and manipulations. The implementation of this approach for spatial data are described in Chapter 5.

4.7 Object-Oriented Subsystems for GIS

Object-oriented GIS may be defined as a spatial system which is based on object-oriented technique being implemented in the system's major components. The

components are built by OO techniques. That is to say there may exist an OO data input module, OO data processing, OO database management, etc. Although there is no standard definition of OO GIS and it is still open to discussion, the term OO GIS is introduced in this work because the subsystems are constructed based on OO techniques. Among the reasons why the term OO GIS is used in this work are the following:

- Object-oriented software engineering (design);
- Object-oriented programming;
- Object-oriented database design and development;
- Object-oriented user interface

are employed in the system development. Aybet(1992) suggests that the application of these four justifies the use of the term object-oriented. Several aspects of OO software engineering and programming are discussed in Chapter 5 (section 5.2). However, briefly, class, method, encapsulation, instances, inheritance and polymorphism provide OO benefits to the users of any implemented system.

Based on the above simple justification for using the term OO, the OO subsystems of a GIS are examined in this work. The developments of these OO subsystems are discussed in Chapter 5.

4.8 Database Management System (DBMS)

A database management system (DBMS) is the software used to interact with the stored data in a database. A database user can perform several common tasks with the data such as display, retrieve, update and manipulate. In general, the aims of a DBMS for the stored datasets are to have the following: links between datasets, data consistency, ease of data access, data security, data sharing, data independency, control and ease of administration (Delobel *et al*, 1995). Links between datasets are established through an

adopted data model. The data model links various datasets in the stored database so that the user can perform the required tasks. Data stored in the database must be consistent with reality as they exist in the real world. How this could be done? A set of rules for maintaining the database is important.

Some OO DBMS features are implemented in this study, namely data retrieval and data query. Populating the database with TIN data and the related data modelling is the main concern of the study. Investigation into other aspects of DBMS (such as updating) can be found in Kufoniyi (1995).

Certain aspects of GIS software components are discussed in the next section.

4.9 Geographic Information System (GIS)

GIS software may have several components as illustrated in Figure 4.10. The possible components are data input, the geo-database, transformation, user interface, display and reporting.

Data input covers all aspect of transforming data captured in the form of, for example, existing maps, text documents, field observations, aerial photographs and satellite images into a compatible digital form (Burrough and McDonnell, 1998). An example operation that was developed in this work is rasterization. The rasterization converts vector data into raster. Format conversions also play an important role in the data input where existing digital data could be converted into a specific format for a particular GIS system.

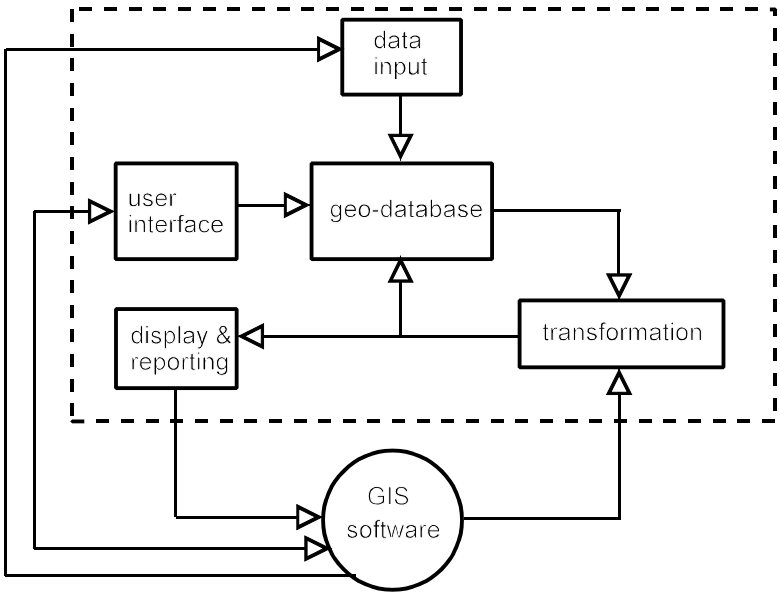


Figure 4.10 Major software components of a GIS (after Burrough and McDonnell, 1998)

A geo-database can be considered to be the digital form of a geo-spatial model which is a replica of some aspects of some portion of the earth's surface. It concerns the way in which data about location, topology, and attributes of geo elements such as points, lines, surfaces and other complex entities are structured and organized. Databases are central to GIS. Data in a database is organized (defined) and manipulated by software known as a Database Management System (DBMS), or more specifically by the Data Definition and Data Manipulation languages within that software.

Display and Reporting (or data output) concerns the way data are displayed and how the results of analyses are reported to the users. Text, tables, maps, and figures are the most common forms of data output. Maps are the most common output from spatial databases, and could be presented either on hardcopy (paper) or softcopy (computer screen) media.

Transformation deals with two categories of operations (Burrough and McDonnell, 1998): (a) operations needed to remove errors from the data or to bring them up to date

or to match them to other data sets, and (b) the large arrays of analysis methods that may be applied to the data in order to achieve answers to the questions asked of a GIS (i.e. queries). Examples of typical transformation operations include geometric computation, map overlay, network analysis, map projection, logical data retrieval, calculation of areas and perimeters. Other kinds of manipulation may be extremely application-specific and their incorporation into a particular GIS may be only to satisfy the particular users of that system (Burrough and McDonnell, 1998). In this study the geometric computations and interpolations on TIN data structures which have been developed may be regarded as operations of data transformation.

A user-interface in GIS supports the interaction of users with the system. In recent years, this aspect has received a considerable amount of attention in GIS research and development. The simplest forms of user-interface for GIS are menu-driven commands that can be selected by simply pointing and clicking with a mouse, and this is an efficient way of providing complex functionality for ordinary users (Burrough and McDonnell, 1998). Other types of user-interface could be in the form of bit-mapped displays, windows, menus, dialog boxes, icons, direct-manipulation, document-centric metaphor (Khoshafian and Abnous, 1995). Many new concepts and techniques exist, and more are becoming available such as Windows, Multi Document Interface, Document/View, Tools-bar, Status-bar, and Icons. Frameworks for creating these user interfaces are available in almost all major programming languages. In this study, an OWL framework of the Borland C++ compiler, called Object Window Library (OWL) has been utilised to create a simple interface.

4.10 The OO TIN GIS

OO TIN GIS is based on several fundamental concepts and aspects of spatial data which have been discussed in the previous sections and chapters. The basic components in the system are data input processing, TIN data construction, TIN database, transformation operations, data output, and user-interface. The system takes the components in Figure

4.10. In this study rasterization forms the major operation in the data input module. Figure 4.11 shows the other major components of the proposed system which includes the use of other commercial software, i.e. ILWIS and AVS. These two packages are used for display purposes only. Some of the operations such as rasterization and other TINs data construction computations are validated visually. In this study a simple user interface as part of the software development was also built.

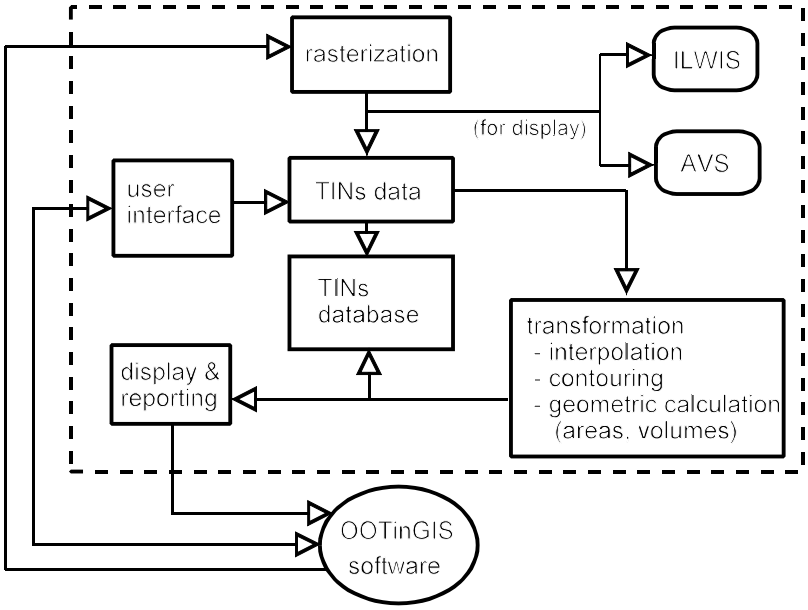


Figure 4.11 The proposed system for the TIN-based spatial data

Beside the programs developed in this work for databasing purposes, a commercial database package, called POET OODBMS has been utilised. The DBMS package is used for the development of OO TIN spatial data. Detailed discussion on the implementation of these components and the related functions are discussed in the Supporting Algorithm chapter (Chapter 6), and also in the Implementation and Test chapter (Chapter 7).

4.11 Summary

This chapter introduces the OO TIN GIS, a system based on the object-oriented approach and TIN spatial data. Spatial data modelling for TIN data, the relevant aspect of data structuring, databasing for such data and the user interface have been described. These all become part of the major building blocks for a geo information system. Several fundamental concepts and aspects of spatial data modelling and how they contribute to the development of geo information system software (i.e. from concepts to a system) have been discussed. All these concepts are implemented and described in chapters 5, 6 and 7.

Chapter 5

Object-Orientation of TIN Spatial Data

5.1 Introduction

The capabilities of object-oriented (OO) techniques have in recent years presented a very promising tool for the development of information systems, especially those requiring the implementation of complex data modelling. OO programming techniques are now being applied widely. OO programming has tremendous potential; GIS is one example. OO techniques of programming and design promise to produce more easily maintained software for less effort and expense (Ross *et al*, 1992). Conventional software development suffers from a number of drawbacks such as endless lines of code, while OO programming allows programmers to build an application program by using existing or easy-to-build entities called objects (object - the term used in OO programming for an instance of a constructed class). Therefore, it seems natural to apply OO techniques for geo-scientific computations such as TIN spatial data modelling.

This chapter provides descriptions of TIN tessellations and spatial data modelling using OO techniques. OO concepts are discussed in section 5.2. OO design for TIN tessellations is discussed in section 5.3. A discussion on the development of TIN spatial data modelling is provided in section 5.4, and the POET OO DBMS development is in section 5.5. The development of OO TIN-based systems for GIS is discussed in section 5.6 and then followed by a summary of this chapter. Brief descriptions of the classes are provided in this chapter whereas the detail of each class is presented in Appendix B and C. The implementation of OO techniques for TIN data tessellations has been further discussed in Abdul-Rahman (1999). Further implementation using an OO database management system (DBMS) is described in Abdul-Rahman and Drummond (2000).

5.2 Object-Oriented Concepts

Object-oriented conceptual modelling is now widely utilised in many fields including GIS. The concepts of OO such as object classification, encapsulation, inheritance, and polymorphism are able to ease the modelling of complex real world objects. As mentioned above the object-oriented approach is now being promoted as the most appropriate method for modelling complex situations that are concerned with real-world phenomena, and thus applicable to GIS. Object-oriented concepts are considered more flexible and powerful than the traditional structural programming and other major database models such as the relational or entity-relationship model. Object-oriented concepts contribute to modelling as follows:

(a) Considering objects and abstraction mechanisms (classification, generalisation, aggregation and association), these aspects of OO can be used for modelling real world phenomena, e.g. modelling of spatial data for geoinformation systems; and

(b) Considering inheritance, propagation, encapsulation, persistence, Abstract Data Type (ADT), polymorphism and overloading, these aspects of OO can be used to construct and implement the model discussed in (a).

The usefulness of these concepts in spatial modelling is explained below.

5.2.1 The abstraction mechanisms

Data abstraction is a method of modelling data. Object-oriented design uses four major abstraction mechanisms: (1) classification, (2) generalization, (3) inheritance, and (4) polymorphism. In object-oriented programming, any physical or logical entity in the model is an "object". The definition of a type of object is called a "class", and each particular object of that type known as an "instance" of the class. Once a class has been defined, it can, potentially be reused in other programs by simply including the class definition in the new program. However, it is not necessary for the programmer who

uses a class to know how it works, they simply need to know how to use it. The definition of operations on or between objects are called "methods", and the invocation of methods is referred to as "passing a message". Recent research in software engineering has promoted an object-oriented design method by which real world objects and their relevant operations are modelled in a program which is more flexible and better suited to describe complex real world situations (Khoshafian and Abnous, 1995). Object orientation also may be considered as a particular view of the world which attempts to model reality as closely as possible (Webster, 1990). Details of all relevant OO concepts (object, abstraction, data types, class hierarchy, inheritance, classification, aggregation, generalization and association) can be found in the OO literature such as Booch (1990), Bhalla (1991), and Stroustrup (1997). The following are some OO terms:

Classification

Classification can be expressed as the mapping of several objects (instances) onto a common class. In the object-oriented approach, every object is an instance of a class (a class is a fundamental building block in an OO language). It describes common features of a set of objects with the same characteristics; a class also defines nature of a state and behaviour, while an object records the identity and state of one particular instance of a class. Abstract Data Type (ADT) is the name of the mechanism to create a class of spatial objects or any class in a domain of objects. An object is a basic run-time entity in an object-oriented system. This entity includes data and procedures that operate on data. Viewed from a programming stand point, objects are the elements of an OO programming system sending and receiving messages.

Generalisation

Generalisation in OO provides for the grouping of classes of objects, which have some operations in common, into a more general superclass. Objects of superclass and subclass are related by an *"is a"*- relation, since the object of a subclass also "is-a" (an) instance of a superclass.

Inheritance

Inheritance allows the building of a hierarchy of types or classes that best describes the real world situation in the application field. Each class can take all or part of the structural or behavioural features from other classes, which are its parents. In turn, the newly defined class is a child of the classes from which it has inherited its features. Inheritance helps in deriving application-oriented classes without starting every definition from scratch. Also, it makes it easier to create logically complex classes from simpler classes.

Polymorphism

Polymorphism is a mechanism to define the different actions of the same named function on different classes. It is implemented by inheriting some functions from parent classes and overriding or modifying part of them. Usually, the newly created class has similar but not the same behaviour as its parents for that functional aspect. Polymorphism provides great flexibility in class derivation, for example, the *calculate_perimeter* operation may have different implementations for different classes such as class "area", class "triangle", class "polygon", etc. Each class performs the calculate perimeter operation differently although it has the same function name.

5.2.2 The programming language

Object-oriented concepts were originally developed in early programming languages such as Simula in 1960's. Other OO programming languages such as Smalltalk, C++ and Java have also been developed since then. Although Java is said to be widely used for the Internet or distributed computing environment these days, the C++ language is much more widely used and offers more OO concepts than other languages (Stroustrup, 1997). There are several C++ compilers available from major software/compiler vendors for a wide variety of computer systems. Most of these compilers are meant for a wide variety of scientific computing tasks, including for instance geoinformation

modelling and computations. In the work reported in this thesis, the Borland™ C++ compiler was utilised for all the software development.

5.3 Object-Oriented TIN Tessellations

OO TIN tessellation software has been developed for the construction of 2D and 3D TIN data structures. The algorithms are described in Chapter 6. The descriptions of the OO TIN tessellations follow (section 5.3.1).

5.3.1 Classes for 2D TIN tessellations

Using the above OO mechanisms, the spatial tessellations are designed as shown in Figure 5.1. In this design, the Booch (1990) notation was used to represent the hierarchy of the classes. Booch has provided one of the techniques for designing class hierarchy. Other possible techniques are notations such as those of Rumbaugh and the Unified Modelling Language (UML). In the two-dimensional (2D) spatial tessellation, four major classes have been recognised, the classes are *TDistanceTransformation* (TDT) class, *TVoronoiTessellation* (TVor) class, *TTinGeneration* (TTinGen) class, and *TTinView (*TTinView) class, see Figure 5.1. The TDT class is used to calculate and generate a distance transformed image of given object pixels. The *TVor* class is used to generate the Voronoi image of the object pixels. The corresponding TIN of the object pixels can be determined by using the *TTinGen* class, and the TIN viewing is handled by the *TTinView* class.

In this work, not all OO mechanisms were used. The two most useful mechanisms are classification, and inheritance. The following sections describe all the relevant classes associated with spatial tessellations.

The Class *TDistanceTransformation* generates a distance transformed image from a given rasterised data set. Operations or methods in this class are: *SetBackground*, *GetUpperMask*, *GetLowerMask*, *ForwardPass*, and *BackwardPass.* The details of these

methods or procedures were fully described in Abdul-Rahman and Drummond (1998, 1999). Here, only their relationships with other classes in the class hierarchy are described. The details (class headers which includes all the related attributes and methods) for each class are represented in Appendix B.

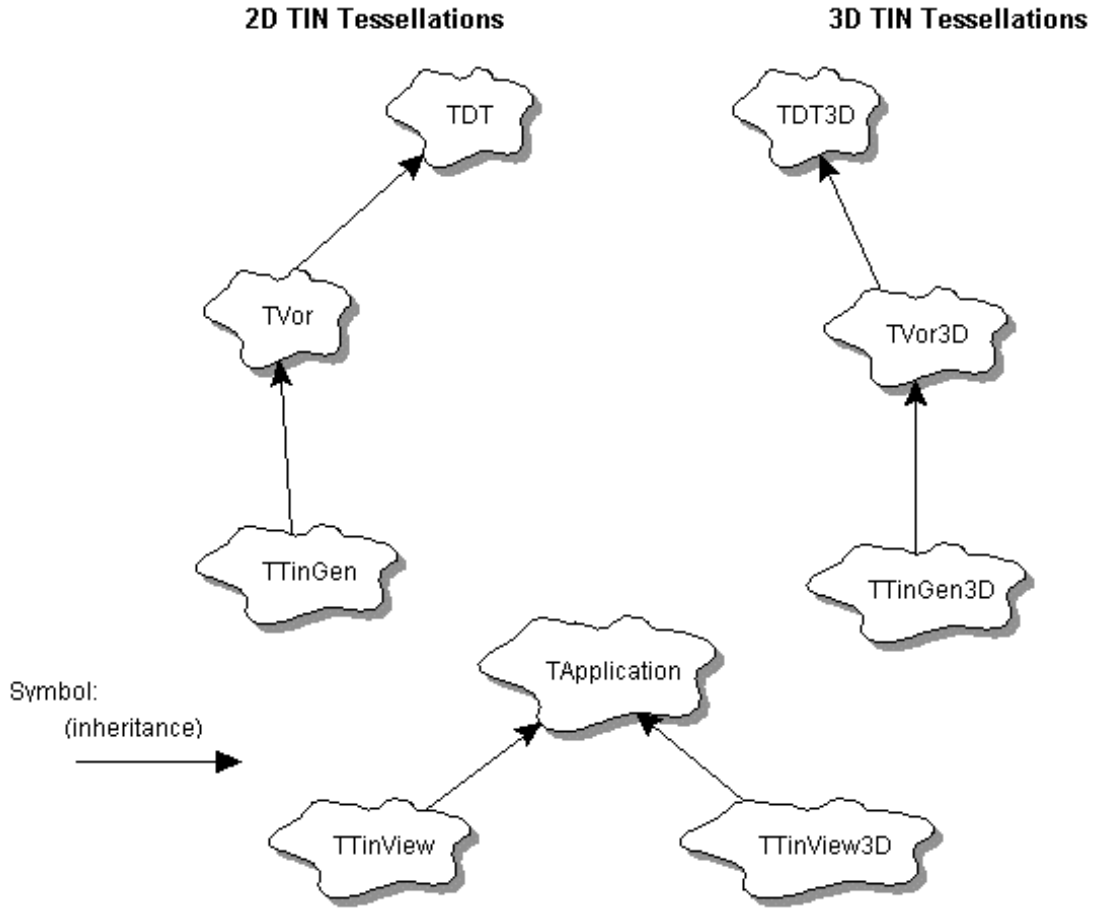


Figure 5.1 The classes hierarchy for the 2D and 3D TIN tessellations

The following class *TVoronoiTessellation* generates a Voronoi image from a given distance transformed data set. The major methods in this class are *ForwardVoronoi* and *BackwardVoronoi*. These two operations were to generate the tessellated image in two passes. The forward pass begins from the top left corner of the image while the backward pass works reversely (i.e. from the bottom-right pixel to the top-left pixel). The class mentioned above, *TTinGeneration* produces a TIN from a given Voronoi image data set. The *ScanlinesUp* and *ScanlinesDown* methods are to detect the TIN's triangles

from the Voronoi images. After having generated the TIN then, the next task is to display (visualize) them. The visualization make uses of the Borland's C++ compiler predefined class *TApplication*, that is the superclass for the TTinView

- The following gives the definitions of the 2D TIN classes:

```
class DistanceTransform                    // declaration of the DT class which contains
    {                                          the data structure, and several
                                               methods associated with the DT
                                               operations.
    public:
    // member data
    typedef struct MpiStruct
    {
    short Nscanlines;
    short Npixels;
      ...
     } MpiType;

     // member functions
    DistanceTransform();        // constructor

    void SetBackground(ImagePPtr Pixel, int Bg, int Fg);
    void GetUpperMask(int r, int c, ImagePPtr Pixel, Mask& MaskPix);
    void GetLowerMask(int r, int c, ImagePPtr Pixel, Mask& MaskPix);
    void DistancePassOne(ImagePPtr Pixel);
    void DistancePassTwo(ImagePPtr Pixel);
    void ForwardDistance();
    void BackwardDistance();
    ~DistanceTransform();       // destructor
    };

class TVoronoiTessellation : public TDistanceTransform
    {
    public:
    ...
     // Function members
    void CopyImage(ImagePPtr, ImagePPtr&);
    void SetBackground(ImagePPtr, int, int);
    void GetUpperMaskDist(int, int, ImagePPtr, Mask&);
    void GetLowerMaskDist(int, int, ImagePPtr, Mask&);
    void GetUpperMaskVoronoi(int, int, ImagePPtr, Mask&);
    void GetLowerMaskVoronoi(int, int, ImagePPtr, Mask&);
    void ForwardPass(ImagePPtr, ImagePPtr);
    void BackwardPass(ImagePPtr, ImagePPtr);
    void ForwardVoronoi();
    void BackwardVoronoi();
     };

class TINGeneration
    {
    public:
    typedef struct MpiStruct
```

```
    {
    short Nscanlines;      // no. of image rows
    short Npixels;         // no. of image columns
     ...
    } MpiType;

typedef struct VertexStruct
    {
    DataType N1;
     ...
    } TVertex;

typedef struct TsNodeStruct
    {
    short x;
     ...
    } TsNode;

// function members
void GetMask(int, int, ImagePPtr, Mask&);
void GetSubImage(int, int, ImagePPtr, Mask&);
void ScanlinesUp(Mask);
void ScanlinesDown(Mask);
void Scanlines(ImagePPtr);
void MakeTIN();
 ...
};
```

5.3.2 Classes for 3D TIN tessellations

The tessellations of the 3D TIN have been developed; their class hierarchies are very similar to the 2D TIN version. The only difference is the computation dimension (the additional third dimension), and the way to visualize the generated 3D tessellation files. The 3D tessellations also have four major classes. The classes are *TDistanceTransformation3D* (TDT3D), *TVoronoi3D* (TVor3D), *TTinGeneration3D* (TTinGen3D), and *TTinView3D* (TTinV3D), recalling Figure 5.1. Detailed definitions of each class are presented in Appendix B.

For purpose of displaying the DT and the Voronoi images, the ILWIS™(1996) and AVS™(1997) packages have been utilised. The latter package is for the 3D images whereas the former package is for the 2D images.

5.4 Object-Oriented TIN Spatial Data Modelling

In this section we provide a discussion of the OO TIN spatial data modelling techniques. The general modelling steps (recalling Figure 4.2 of section 4.3) can be considered to describe TIN spatial data modelling. That is, the three-step approach, namely the conceptual, the logical and the physical steps. In this work, the conceptual step is implemented by utilising TIN as a method to represent spatial objects. Spatial objects are perceived as TINs. Then, by having the TINs data constructed, a model to describe the objects (i.e. their connectedness between objects) can be established. The description of the objects and how they relate to each other, for example from TIN nodes to TIN surfaces is the logical step. All the OO techniques created classes (Node, Edge, Area and Body) are then physically modelled in the OO database environment.

The class schema for spatial data modelling are described below.

5.4.1 The class schema

The schema is based on several classes, they are Spatial Objects (the super class), and four major subclasses which are Node, Edge, Polygon, and Solid.

Spatial objects

The spatial object class is a general class of the real world objects. It is the super class in the class hierarchy. In this work, an assumption is made that all other objects are derived from the superclass (TSpatial object), see Figure 5.2. All terrain objects could be categorised into several sub classes such as points, lines, areas, and solids (volume) features. In OO modelling, these feature types are the classes in the modelling hierarchy.

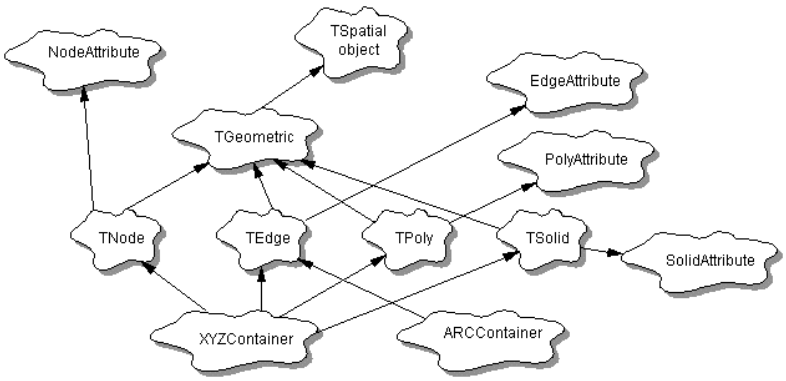


Figure 5.2 The class diagram (using the Booch notation)

Node

A node can be considered as the most basic geometrical unit in spatial data modelling. It may represent point entities or point objects at a particular mapping scale. Examples of point objects are wells, terrain spot heights, and the like. In geoinformation, we may represent these objects by a class called a node class. The coordinates of the nodes (including the nodes represent edges) are held by a coordinates container class, called XYZContainer class.

Edge

An edge can be represented by one node at each end (i.e., a start node and end node). In this study we consider two end points make a straight edge. This edge type can be used to represent linear features. The arc container class, called ARCContainer holds all the arcs. The arc containers also serve any other class which requires arcs data in their operations, for example the polygon class needs the arcs in order to form polygons.

Polygon

A polygon (sometimes known as a surface) is used to represent area features such as lakes, ponds, etc. A polygon may be constructed by chains of closed edges.

Solid (or Body)

A solid is a representation for solid or body features such as buildings, trees. A chain of points, lines and surfaces form body objects. A 3D TIN can be represented by a series of triangle nodes and edges as mentioned in the previous chapter.

The class schema in Figure 5.2, depicted using Booch (1990) notation is the representation of the TIN spatial data model. The schema has four geometric classes namely *TNode, TEdge*, *TPoly*, and *TSolid* and two types of containers: geometry and attribute. The geometric containers contain the XYZ locations (held by the *XYZContainer*) and the *ARCContainer* whereas the attribute containers are for the thematic values, e.g. names. The attribute information is held by the *NodeAttrbute,* the *EdgeAttribute, PolyAttribute,* and *SolidAttribute.*

The following gives the definitions of the classes as presented in Figure 5.2. In Booch notation, each class is represented by a "cloud-look" diagram. It contains data and methods for a particular class. The arrow shows the link between a class with another classes. More detailed class definitions are given in Appendix C.

- The geometric classes (written in C++ style) are:

```
class TNode
    {
    public:
            struct XYZContainer                    // this is the XYZContainer struct

            {
            double x;
            double y;
            double z;
            };
    XYZContainer Point[maxpoint];

    struct NodeAtrContainer                        // this is the NodeAtrContainer struct
    {
    int NodeNum;
    string NodeName;
    };
    NodeAtrContainer NodeAtr[maxnodename];
```

```
...
...

void GetXYZCoordinates();                    // the methods for the class follows
void Get2Node();
void NodeAttribute();
};
```

The class TNode describes the structure for the XYZContainer, the NodeAtrContainer as indicated by the 'struct' keyword. The class also has several methods such as GetXYZCoordinates(), Get2Node(), and NodeAttribute().

The following TEdge class has several methods as indicated after the keyword void, int and float.

```
class TEdge
    {
    public:
            ARCContainer Arc[maxarc];                    // the declaration for the
                                                         containers follows.
            EdgeNameContainer-  EdgeAtr[maxarcname];
            ...
            ...
            void ReadARCs();                             // the associated methods
                                                         follows.
            void GetArcLength();
            void GetArcAttribute();
            int CheckQuadrant(float, float);
            float Bearing(float, float, float, float);
            float GetArcAzimuth(float, float, float, float);
            void EdgeAttribute();
    };
```

For the class TPoly, it has TINContainer structure, TENContainer structure, and several related methods for the polygon (areal) objects.

```
class TPoly
    {
    public:
    struct TINContainer                              // the declaration for the
                                                     TINContainer struct.
    {
            int Node1;
            int Node2;
            int Node3;
    };
    TINContainer Triangle[maxtriangle];

    struct TENContainer                              // the declaration for the
                                                     TENContainer struct.
            {
            int TriNum;
```

```
            int NumofNbr;
            int Nbr1;
            int Nbr2;
            int Nbr3;
            };
    TENContainer TINNbr[maxtriangle];

    void ReadTINs();                                    // the associated methods
                                                        follows.
    void GetTINNeighbour();
    void GetTINNodes(int,
            double&, double&, double&,
            double&, double&, double&,
            double&, double&, double&);
    float GetTINArea();
    void GetPolyArea(int, int, double&);
    };
```

Whereas the class TSolid contains TENContainer struct, TINContainer struct, and the associated methods for the operation on the solid objects. Their corresponding declaration in C++ follows:

```
class TSolid
    {
    public:
    struct TENContainer                                 // the declaration for the
                                                        TENContainer struct.
            {
            int Node1;
            int Node2;
            int Node3;
            int Node4;
            };
    TENContainer TEN[maxtriangle];

    void ReadTENs();                                    // the associated methods for
                                                        the class TSolid.
    void Get3TINNodes(int,
                    double&, double&, double&,
                    double&, double&, double&,
                    double&, double&, double&,
                    double&, double&, double&);

    float GetVolume(double, double, double,
                    double, double, double,
                    double, double, double,
                    double, double, double);
    };
```

The schema also describes several classes of geometric containers. The classes are XYZContaner, TINContainer, TENPolyContainer, TENContainer, ARCContainer. The declaration of these classes follows:

```
class XYZContainer                          // the class contains (x, y, z) coordinates
    {
```

```
    public:
    double x;
    double y;
    double z;

    XYZContainer() {}
    ~XYZContainer() {}
    };

class TINContainer                    // the class contains the 3 nodes of the TIN.
    {
    public:
    int Node1;
    int Node2;
    int Node3;
    };

class TENPolyContainer                // it contains the neighbouring topology for the
                                      TIN.
    {
    public:
    int TriNum;
    int NumofNbr;
    int Nbr1;
    int Nbr2;
    int Nbr3;
    };

class TENContainer                    // it contains the 3D TIN's 4nodes.
    {
    public:
    int Node1;
    int Node2;
    int Node3;
    int Node4;
    };

class ARCContainer                    // the two end nodes for arcs.
    {
    public:
    int StartNode;
    int EndNode;

    ARCContainer();
    ~ARCContainer();
    };
```

The following describes the two classes for the attribute data. The classes are EdgeNameContainer, NodeNameContainer.

- The attribute classes are:

```
class EdgeNameContainer
    {
    public:
```

```
    int EdgeNum;
    char EdgeName[30];
    };

class NodeNameContainer
    {
    public:
    int NodeNum;
    char NodeName[30];
    };
```

5.5 Object-Oriented TIN Spatial Database Development

This section explains the development of an OO database for TIN data using a commercial database management system, that is POET OO DBMS.

5.5.1 The POET OO DBMS

The POET (Persistence Object and Extended Technology) DBMS is utilised in this work as part of the OO spatial data modelling. The package works under Windows 95 operating system for the PC environment, that is the major computing environment adopted in this work. The package is also chosen due to it economic reason, it cost less and can work with the Borland C++ programming language, the language adopted by the author for the entire software development in this work. The database package is said to have the following capabilities (POET, 1996) including:

a. Encapsulation;
b. Inheritance; and
c. User-defined data types.

Which are among important OO features of POET that can be useful for the TIN spatial data modelling purposes.

5.5.2 The POET database schema

The DBMS is used to generate the OO database from the constructed TIN spatial data.

In this work the schema needs to be modelled according to the POET database model (POET, 1996), that is it is required that all the C++ classes are constructed as classes which POET can understand. In this case, all the classes in the schema have to be compiled by the POET PTXX compiler as shown in Figure 5.3. The PTXX compiler maps all the normal C++ classes into the several relevant PTXX schema files which in turn are used for writing application programs (running under the normal C++ compiler) as well as for populating the database. The PTXX compiler also generates the OO database from the schema.

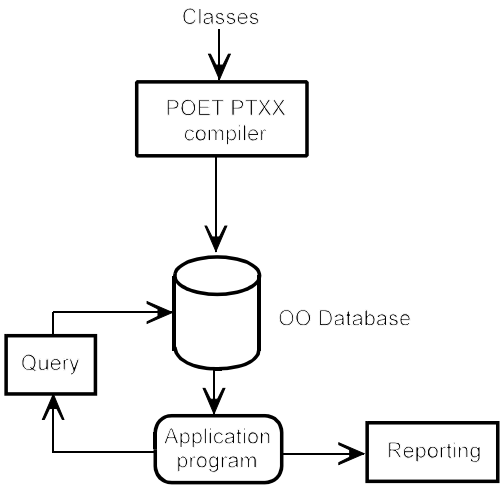


Figure 5.3 The POET database development flow

5.5.3 The POET database browser

In POET, once the database schema has been properly compiled, then the generated database can be inspected by using the built-in browser. All generated objects can be examined for further database operations. Figure 5.4 illustrates the screen shot of the POET Developer module where the TIN database is developed.

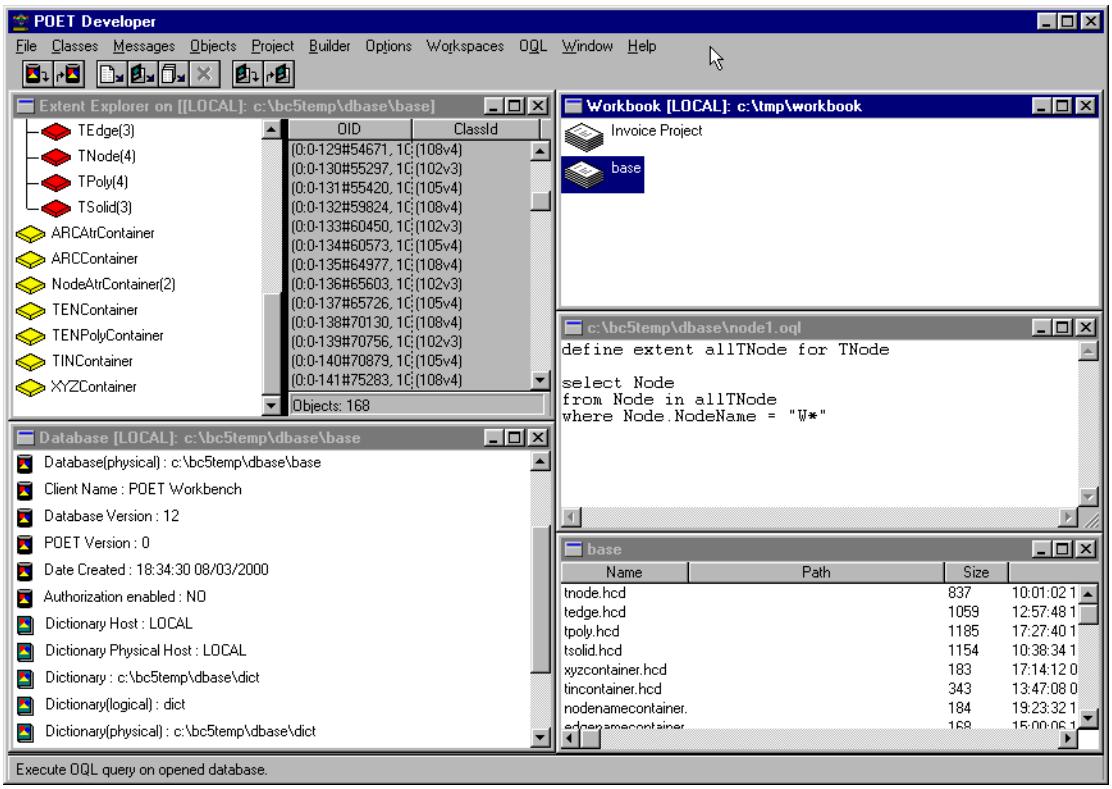


Figure 5.4 The POET Developer which was used to develop the TIN OO database and support database retrieval(query).

5.5.4 POET database query

The generated database can be queried by using a built-in database query facility within the POET Developer module. This built-in technique is adopted for this work. A query language similar to structured query language (SQL) can be utilised. The language is called Object Query Language (OQL) and detailed syntax of the language can be found in POET(1996). The following is an example of a query which can be performed from the database:

```
defined extent allTEdge for TEdge;
select Edge
from Edge in allTEdge
where Edge.EdgeAtr.EdgeName = "River*"
```

In order to be able to manipulate the database, an application program has been

developed. This program runs under the normal Borland C++ compiler but it makes use of the files which were generated by the POET PTXX compiler. Appendix J (Volume II of this thesis) gives a detailed listing of the POET-compatible program.

5.6 Object-Oriented TIN-based Subsystems for GIS

5.6.1 The subsystems

The OO TIN GIS is based on several fundamental concepts and aspects of spatial data which have been discussed in the previous sections. Basic components of the system are data input processing, TIN data construction, TIN database, transformation operations, data output and user-interface. Rasterization forms a major operation in the data input component. Figure 5.5 shows the other major component of the proposed system - visualization, which uses the commercial software, i.e. ILWIS™ (Integrated Land and Water Information System) and AVS™ (Advanced Visualization System). These two packages are only used for display purposes, more especially for validating the output from the rasterization process. A simple user-interface as part of the software development is also developed. Besides the programs written by the author for databasing purposes, a commercial database package is also utilised, called POET™ OODBMS as mentioned. The DBMS package is for the development of the OO TIN spatial database.

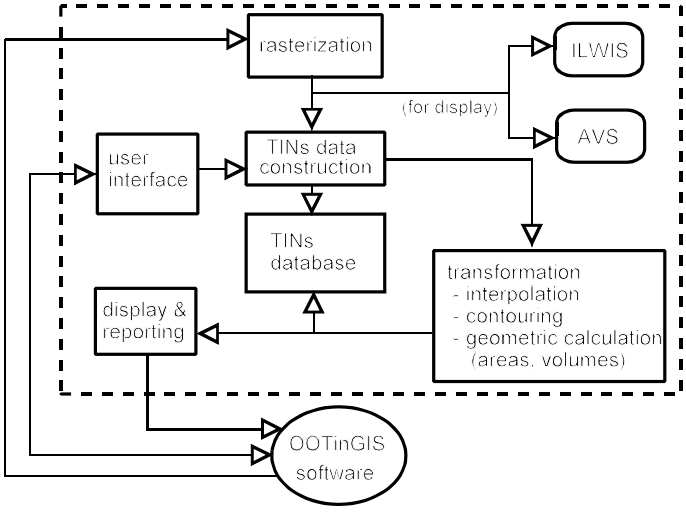


Figure 5.5 The proposed system for the TIN-based spatial data

5.7 Summary

This chapter introduced the implementation of object-oriented techniques for TIN (2D and 3D) spatial data. The chapter revealed the usefulness of a commercial OO DBMS package to develop TIN spatial database schema as described in section 5.5. The software development which has been implemented in this chapter could be applied to a much larger system built from those subsystems.

2D and 3D TIN tessellation is one of the major components of this research. These tessellations are shown to work perfectly in the OO environment. The approach which was implemented in this chapter can be used for developing a TIN-based GIS system. The graphic output of the tessellations shown in Figure 5.6 clearly demonstrates the workability of the OO technique. More results of the implemented subsystems are presented in Chapter 6.

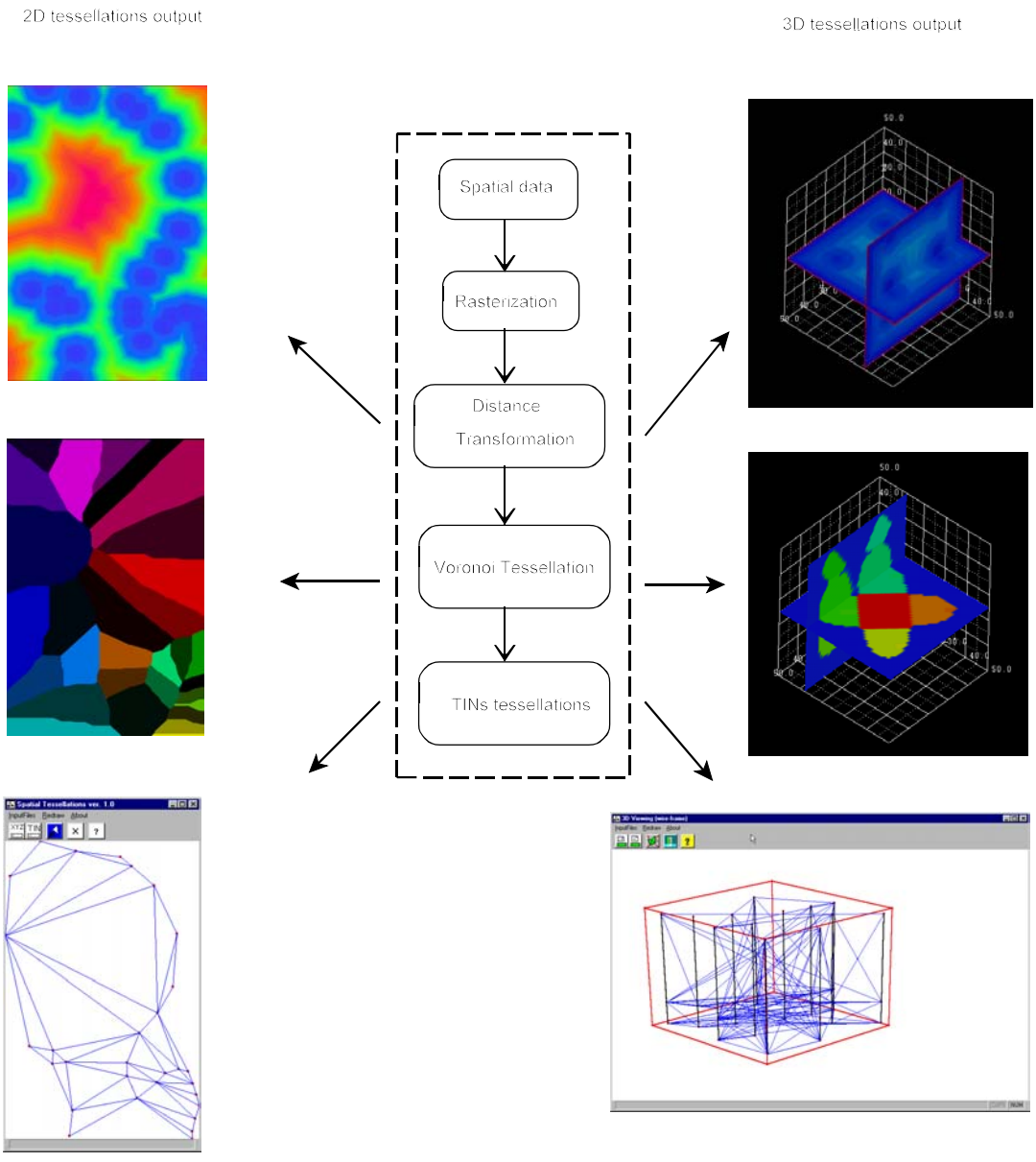


Figure 5.6 The 2D and 3D TIN tessellations

Chapter 6

The Supporting Algorithms

6.1 Introduction

This chapter introduces several major algorithms for TIN spatial data structuring and constructions. Data structuring for terrain surfaces has been investigated for several decades. The main concern of the earlier investigations was the suitability and the adaptability of data structures for representing terrain surfaces. In the late 1970's a triangular irregular network (TIN) data structure was presented (Peucker *et al.*,1978). Several methods and techniques have emerged for the construction of TIN structures (McCullagh and Ross, 1980; Watson, 1981; Mirante and Weingarten, 1982). Most of the techniques were attributed to Delaunay (1934) and known as Delaunay triangulation. TINs could be constructed either in the vector or in the raster domain. In this research a raster technique for the construction of the TINs (2D and 3D) is used.

In this chapter the algorithms for the construction of 2D TIN and 3D TIN spatial data are introduced. These algorithms have the names: Distance Transformation (DT), Voronoi Tessellations, Triangulations, and Triangulations Data Structuring. In this work visualization and rendering routines for 2D and 3D data have also been developed, as have rasterization programs for TIN data construction purposes. Each algorithm is explained in detail together with its C++ pseudo-code. Detailed codes are presented in Appendix J (in Volume II).

6.2 Distance Transformation

Originally, the term distance transformation (DT) was used by Rosenfeld and Pfaltz (1966) and later by Borgefors (1986). The DT was used to describe an operation of

converting binary images to a grey-level image where all pixels have a value corresponding to the distance to the nearest feature (or object) pixel. The same principle had also been applied in other areas of interest such as raster-based GIS and remote sensing (Gorte and Koolhoven, 1990). The DT provides a method for calculating the distance from every non-object element in a two-valued raster data set to the nearest object element of a set of object elements. The Borgefors DT technique was a fundamental step in this raster-based TIN development. The transformed image can be used to generate a Voronoi tessellated image, and then a set of triangles can be generated from that Voronoi tessellated image. Triangles generated from Voronoi polygons are sometimes known as the dual product of the Voronoi polygons (Fortune, 1992). Borgefors (1986) investigated several types of DTs. These were known as City block, Chessboard, Octagonal, Chamfer 3-4, Chamfer 5-7-11, and Euclidean. Each DT produces different output images and requires a different computation time. Borgefors suggested that Chamfer 3-4 can be used for generating distance transformed images due to its processing simplicity. Description of other DTs can be found in Borgefors (1986). It is not the intention of this section to compare all the DTs but rather to explain them and then use the most appropriate one (i.e. that which is relatively easy to implement); a detailed explanation of the DT which was used in the TIN development is described later in this section.

Distance Transformation (DT) is a technique used in the image processing community for a range of applications, one example is zone mapping (Borgefors, 1986). A zone of accumulated distances could be mapped from a rasterised point. This DT concept is used in this research and the technique is one of the fundamental steps in the construction of the triangulation. The task is to generate a distance-transformed image of object pixels. In a raster image, object pixels could be in the form of random points, digitized points, digitized lines, etc.

Figure 6.1 shows an example of several points whereas the DT image of the points are illustrated in Figure 6.2.

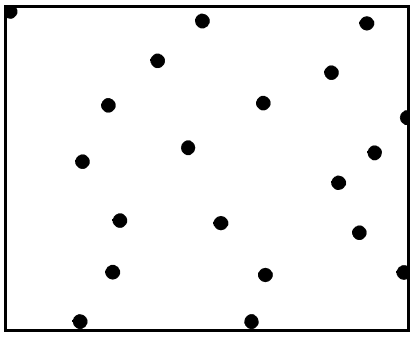


Figure 6.1 Several kernel points (or object points).

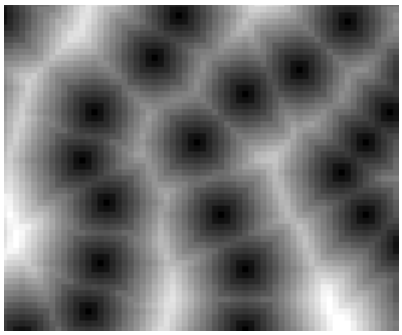


Figure 6.2 The DT image of the several points as shown in Figure 6.1.

In Figure 6.2, the darkest spots represent the location of the kernel points. In the DT, each kernel point is used to generate distance image from neighbouring kernel points. Distances accumulate from the centre of kernel points. In the above example, the centre of kernel points get the value zero (darkest) and the distances gradually increase from the centre (indicated with brighter images) as shown in Figure 6.2. To perform the DT to an image of rasterised points for example, a mask (or a window of 3 x 3 pixels) is required as shown in Figure 6.3(a). The mask has 9 pixels (3 x 3 pixels). This mask is divided into two, called upper mask, and lower masks, as shown in Figure 6.3(d).

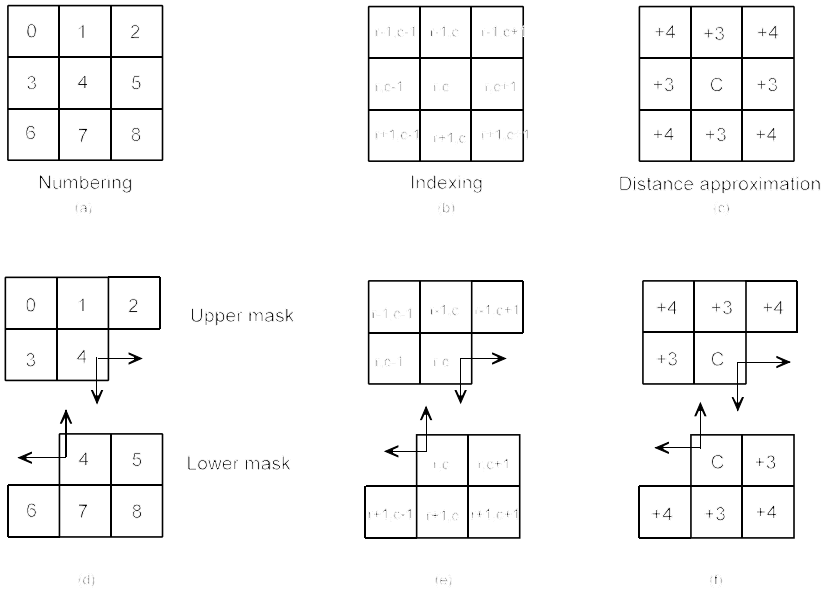


Figure 6.3 Masks for the DT operations

The algorithm works with two passes of the entire image. The first pass (or scan) uses the upper mask while the lower mask is used for the second pass. Each pixel in the mask is indexed according to Figure 6.3(b) where the centre pixel of the mask represents the image pixel then being scanned.

In this algorithm, the DT works as follows: all object pixels are changed to zero (i.e. a value 0) and the rest of the pixels (i.e. the background pixels) to the highest possible value e.g. an integer value of 32767 (of 16-bit data type architecture). The entire image is scanned in two passes using the Chamfer 3-4 mask of the Borgefors DT (Borgefors, 1986) as illustrated in Figure 6.3(c). The first pass (scans with upper-mask) begins from the first pixel (i.e. the top-left pixel) and goes to the last pixel of the image. In the first pass, all the pixels which are covered by the mask get a new value. Each pixel's value has added to it either a value of 3 or 4 depending on its direct or indirect neighbour relationship with the centre pixel (C). Then, the minimum value is determined from the five possible candidates and assigned to the current pixel location. The mask is then moved to the next pixel location. At this next location, the minimum value for this pixel

is again determined and assigned. This process continues to the last pixel location (i.e. the bottom-right pixel) of the image. The result of the first pass operation is used for the second pass which operates in reverse order (i.e. from the last pixel to the first pixel of the image). It is a recursive operation. Finally, a DT image is generated after these two passes are carried out. In this DT image, all pixels contain the approximate distance to the nearest kernel points (object pixels).

The following pseudo-code describes the DT algorithm:

```
// Procedure to Set the background image
void set background()
 {
  Set loop for rows (first row to last row)
   {
    Set loop for columns (first column to last column
     {
      if (Pixel value not equal to background)
          Set Pixel value to zero;
       else
      Set Pixel value to background (highest possible value);
      }
   }
 }

// Procedure to assign the Upper Mask
void GetUpperMask()
 {
  Assign the Mask[0] to Mask[4] to the corresponding pixel locations,
      e.g.,  Mask Pixel[0] = Pixel at [row-1][column-1];
 }

// Procedure to assign the Lower Mask
void GetLowerMask()
 {
  Assign the Mask[4] to Mask[8] to the corresponding pixel locations,
      e.g.,  Mask Pixel[4] = Pixel at [row][column];
 }
```

```
// Procedure to compute distance in forward pass
void ForwardPass()
 {
  Set loop for row(first row to last row)
   {
    Set loop for (first col to last col)
     {
      GetUpperMask();
      If Mask has odd index add 4 to the Mask value;
      else
      Add 3 to the Mask value;
       }
           Get the minimum value of Mask[0] to Mask[4] and assign to this pixel;
     }
   }
 }

// Procedure to compute distance in backward pass
void BackwardPass()
 {
  Set loop for row(from last row to first row)
   {
    Set loop for col(last col to first col)
     {
      GetLowerMask();
      If Mask has odd index add 4 to the Mask value;
      else
      Add 3 to the Mask value;
       }
           Get the minimum value of Mask[0] to Mask[4] and assign to this pixel;
     }
   }
 }
```

The above steps then combined as follows into one main DT routine as follows:

```
// Procedure to compute the distance using forward and backward
void Forward&Backward()
 {
  Reads the input Image;
  Set the Background;
  Compute distance using the FirstPass;
  Compute distance using the SecondPass;
  Write and save the transformed image to file;
 }
```

An image of a DT for a number of points within a data set (kernel points) is illustrated in Figure 6.2. The darkest spots in the image represent the kernel points, and it gradually brightens outward from the points. The DT algorithm appears to work well.

6.3 Voronoi Tessellations

Voronoi polygons are also known as Thiessen or Dirichlet polygons. They have been considered one of the fundamental structures in computational geometry and other fields such as GIS. Voronoi polygons are often used in GIS as a method for analysing points data, for example for finding nearest neighbours (Burrough and McDonnel, 1998). In Voronoi polygons, one centroid point represents one polygon. The extent of each polygon indicates the influence of the centroid point with respect with the neighbouring points.

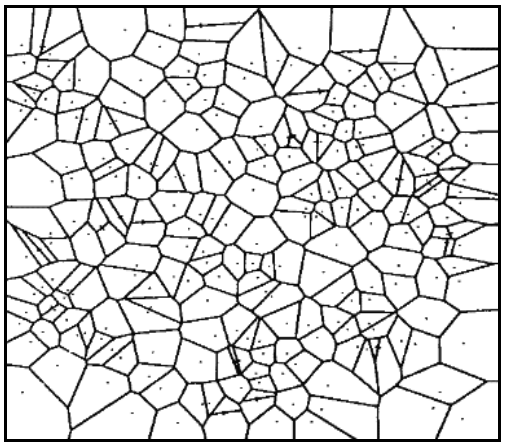


Figure 6.4 Example of Voronoi polygons represented by several data points (after Fortune (1992)).

This type of polygon is useful in GIS as mentioned, e.g. for zone mapping or for determining the region of influence of a phenomenon or buffering (Gold *et al*, 1997). Figure 6.4 shows Voronoi polygons where each is represented by a centroid point.

From the generated DT image of kernel points as described in section 6.2, Voronoi polygons of these points can be constructed. The generation of the polygons can be

done either in parallel or in stages. In this algorithm, the tasks were carried out in parallel. If the DT generation as described in section 6.2 is re-examined, it is seen to involve three steps. First, change the object pixel value to zero (i.e. 0) and the background image to the highest possible value. Second, determine the minimum value of the current pixel location among five possible candidates of the upper mask. Third, assign the minimum value to the current pixel location. In other words, the pixel value represents a distance value of the pixel calculated from the nearby object pixels.

To generate the Voronoi-tessellated image parallel with DT operation, two output files are needed. That is one for the DT image and the other for the Voronoi image. Computing the DT image according to the algorithm describe in section 6.2 involves the following steps at a particular pixel [i, j]: First the mask is "put"on the pre-processed image, the mask centre (having the value 0) at [i, j] of the pre-processed image. Secondly the values of the mask are added to the values pixels that are being covered. Thirdly, the minimum of the 5 resulting values is determined and assigned to [i, j] of the current distance transform image. Before continuing to the next pixel, for which the distance is to be computed, the value for the second output image, the Voronoi tessellation image, at [i, j] has to be assigned. This is done by determining the location of the pixel where the minimum value was found just before, e.g. at [i, j-1]. The pixel value of the original image at [i, j] is then taken and assigned to [i, j] of the Voronoi tessellation image (see Figure 6.5 and also Figure 6.6). This method of computing the Voronoi and DT in parallel was also suggested by Borgefors (1986) with the following quote "the computing of the Voronoi tessellation image can be done by first computing distance transformation from an object pixel while at the same time keep track from which pixel the distance is computed".

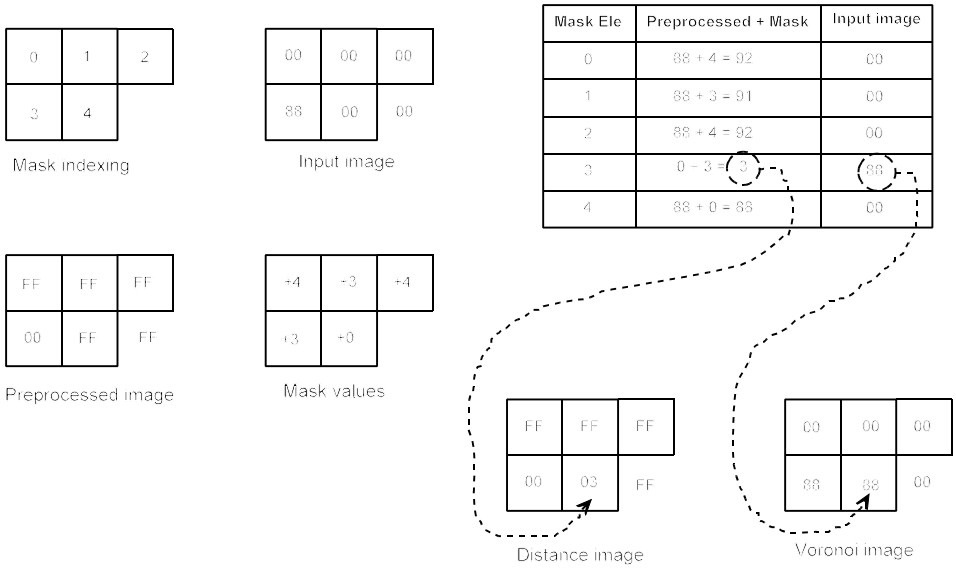


Figure 6.5 DT computation and Voronoi image generation during the forward pass.

A more complete picture for the parallel process of the DT and Voronoi tessellation implementation is illustrated in Figure 6.6 where the outcome of the first pass and the second pass applied to the input pixels is clearly illustrated.

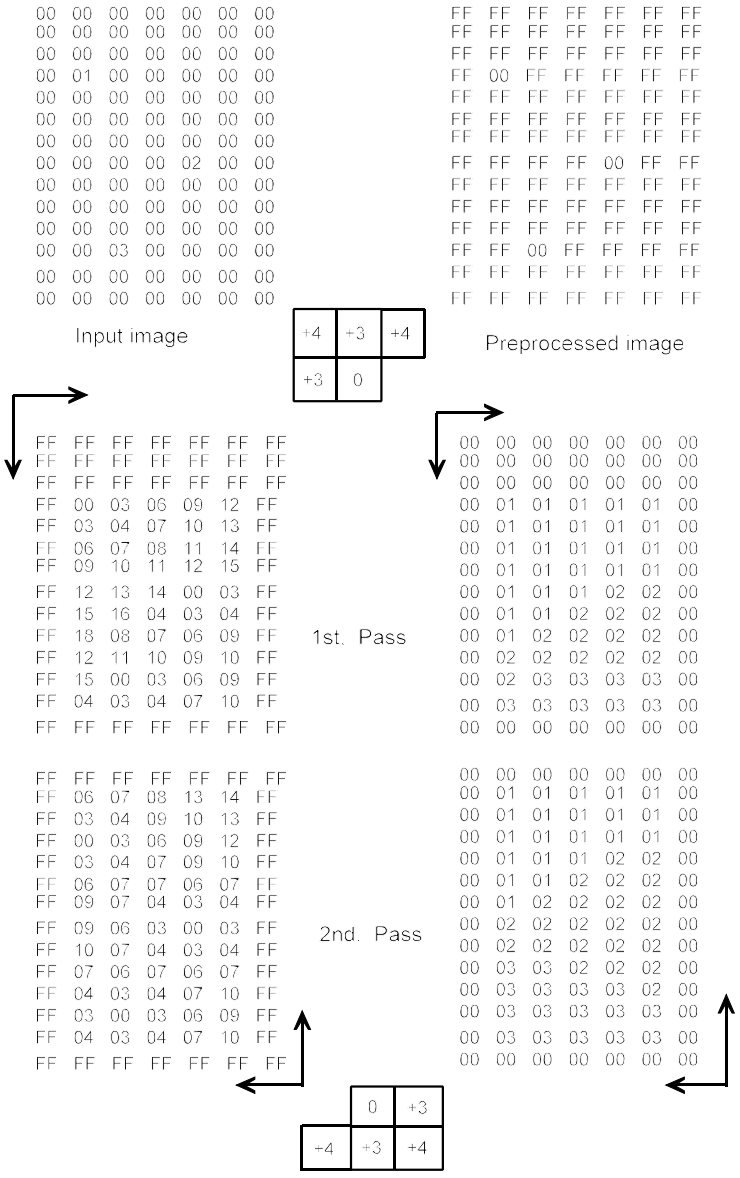


Figure 6.6 DT and Voronoi tessellation parallel computation

The algorithm is tested by using several simulated digitized datasets (Figure 6.7) as well as photogrammetrically captured datasets (Figure 6.9).

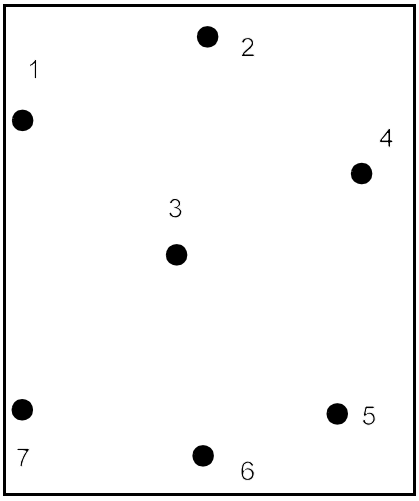


Figure 6.7 Several kernel points

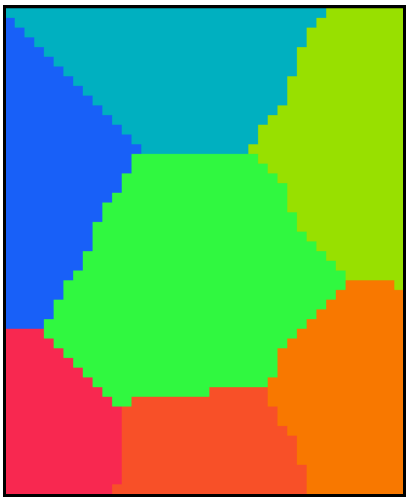


Figure 6.8 The generated Voronoi polygons of the points as shown in Figure 6.7

The Voronoi polygons in Figure 6.8 are clearly delineated. Different image tones represent different polygons as depicted in Figure 6.10 where their kernel points are shown in Figure 6.9.

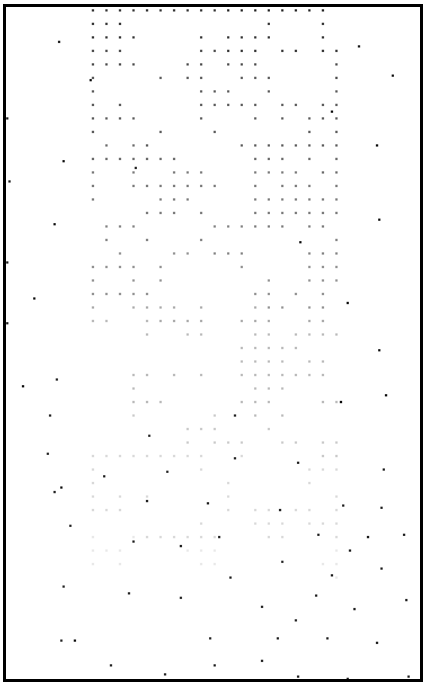


Figure 6.9 The rasterised kernel points of the photogrammetric data sets.

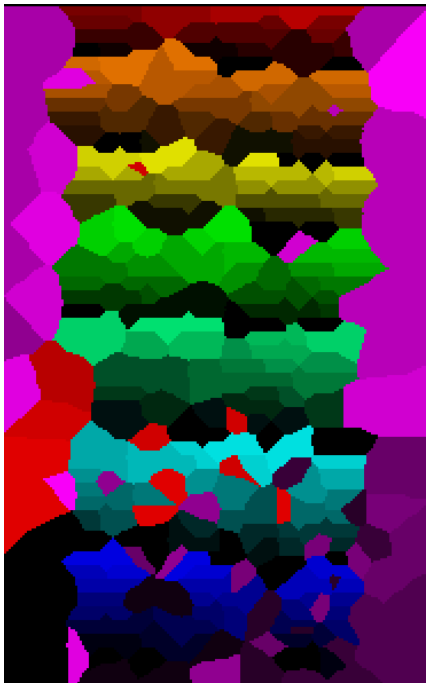


Figure 6.10 The generated Voronoi polygons of the kernel points (Figure 6.9).

6.4 Triangulations (TINs)

Descriptions of triangulations associated with digital terrain modelling (DTM) and surveying can be found in texts such as Petrie and Kennie (1990). A more specific discussion of TIN algorithms for visualization aspect can be found in (van Kreveld, 1997). In this section basic Delaunay triangulation will be described (this method of constructing triangles was attributed to Delaunay (1934)).

The principle of Delaunay triangulation is that the circumscribing circle of any triangle does not contain any point of the data set inside it, see Figure 6.11.

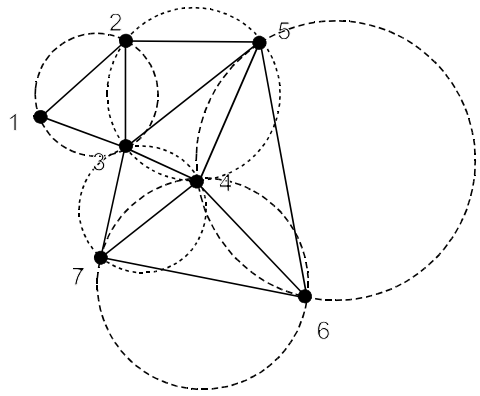


Figure 6.11 Six non-overlapped triangles of 7points created by the Delaunay triangulation technique

A number of triangulation algorithms were developed based on Delaunay triangulations and widely implemented in terrain surface modules in a number of GIS and DTM packages. In such packages, the triangulation is normally known as a triangular irregular network (TIN). Each triangle in a TIN connects three neighbouring points so that the plane of the triangle fits the surface sufficiently. The TIN structure was designed by Peucker and co-workers (Peucker *et al.,* 1978) for digital terrain modelling. As mentioned in the foregoing discussion (recalled section 3.2.3), a TIN is a terrain model that uses a sheet of continuous, connected facets based on a Delaunay triangulation of irregularly spaced nodes or observation points. TIN is considered

to provide a better structure for surface modelling than other structure such as grids (grid for example may not retain the original data). It is not the intention of this section to describe fully the advantages of the structure but rather to briefly mention the TIN primitives instead. The primitives are nodes, lines, and surfaces which were considered the fundamental building blocks for spatial information. This is an interesting consideration from which to develop and implement the TIN package discussed in this work. In two-dimensional space, the 2D TIN can be used for developing a spatial information system, this is because the structure contains spatial data primitives, namely node, line, and surface primitives.

At this point unconstrained triangulations have been developed; that is no other terrain features are incorporated such as breaklines or any linear features except terrain points. A much better triangulation, that is constrained triangulation capable of incorporating such terrain features is discussed in section 6.9. As was mentioned earlier, the triangles in this work are generated using Delaunay triangulation; three kernel points of the neighbouring Voronoi polygons need to be known to form a triangle as shown in Figure 6.12. If there are more than three neighbouring polygons, for example 4 polygons, then there will be two possibilities for triangle formation, see Figure 6.12.

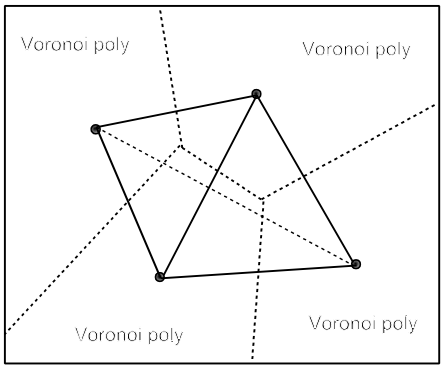


Figure 6.12 The two possible triangles formation

In this work, the triangles are properly constructed, that is the triangles are formed according to the Delaunay technique where there are no ambiguous triangles created.

In other words a correct TIN topology is established. A correct triangle formation can be achieved by searching 3 Voronoi polygon neighbours. In order to find a unique set of 3 points from a Voronoi-tessellated image, a 2 x 2 mask is used (as illustrated in Figure 6.13). The mask is designed to detect only two specific situations where 3 or 4 different pixel values fall inside the mask at a time. These different pixels correspond to the neighbouring Voronoi polygons and the kernel points of these polygons were used to form the triangle. Figure 6.13 shows the mask for detecting the triangle topology.

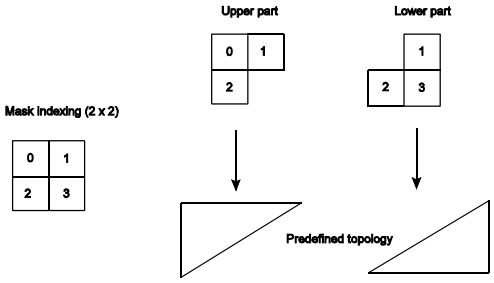


Figure 6.13 Mask (2 x 2) for TIN topology detection

The mask is separated into two parts with the aim of avoiding the overlapping (crossover) triangles, as overlapping triangles are not allowed in the Delaunay triangulation. The mask (2 x 2) is designed to work using a matching operation. The pseudo code for the upper-part mask as follows:

```
if    (mask[0] not equal to mask[1]) and
      (mask[1] not equal to mask[2]) and
      (mask[2] not equal to mask[0]) then
      {
            increase( number of triangles);
            node[0] = mask[0];
            node[1] = mask[1];
            node[2] = mask[2];
      }
```

whereas below is the matching condition for the lower-part of the mask:

```
if    (mask[1] not equal to mask[2]) and
      (mask[2] not equal to mask[3]) and
      (mask[1] not equal to mask[3]) then
      {
            increase(number of triangles)
            node[0] = mask[1];
            node[1] = mask[2];
            node[2] = mask[3];
      }
```

The triangle detection also works with two passes of operations as for the previously discussed DT and Voronoi tessellation operations. The upper-part mask is used to scan the Voronoi image from the first pixel to the last pixel. A triangle is found if four different pixels match either one of the matching conditions imposed by the mask, see Figure 6.14. (Note: the shape of Voronoi polygons do not represent the exact shape. The purpose is to show the concept of triangle detection from generated polygons). The figure illustrates how triangles could be detected.

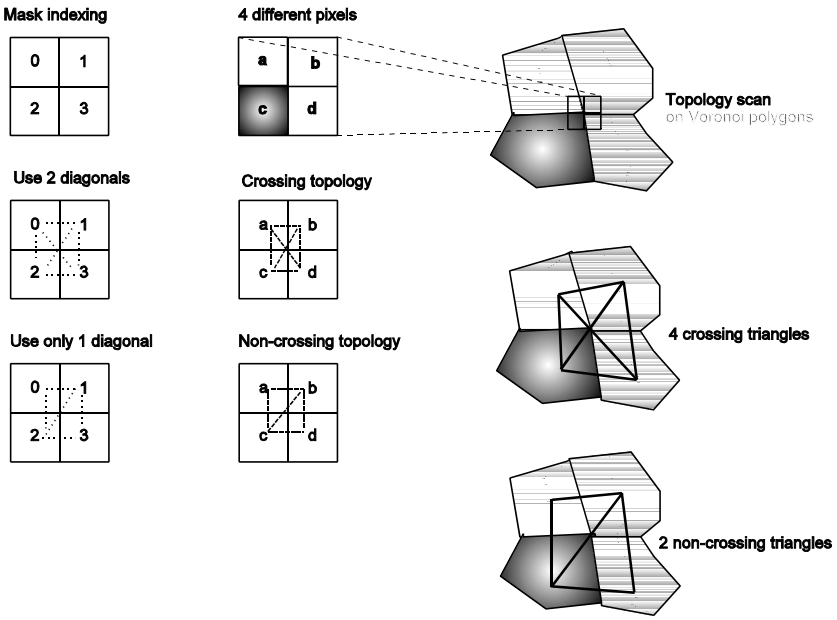


Figure 6.14 Triangle topology detection

The above topology matching condition works only if non-adjacent rasterised points are found in the data set. In other words, two adjacent pixels of rasterised points produce incorrect topology (i.e. a very narrow polygon creates crossing triangles). This situation can happen if one choses an inappropriate pixel size at the rasterising stage of the data sets. Thus to incorporate overcoming this problem, a few lines of conditions were added to the previous matching conditions. The matching condition is as follows:

```
if    (mask[0] not equal to mask[3]) and
if    (mask[0] not equal to mask[1]) and
      (mask[1] not equal to mask[2]) and
      (mask[2] not equal to mask[0]) then
      {
            increase( number of triangles);
            node[0] = mask[0];
            node[1] = mask[1];
            node[2] = mask[2];
            Add triangle to the list;
      }

if    (mask[0] not equal to mask[3]) and
if    (mask[1] not equal to mask[2]) and
      (mask[2] not equal to mask[3]) and
      (mask[1] not equal to mask[3]) then
      {
            increase(number of triangles)
            node[0] = mask[1];
            node[1] = mask[2];
            node[2] = mask[3];
            Add triangle to the list;
      }
```

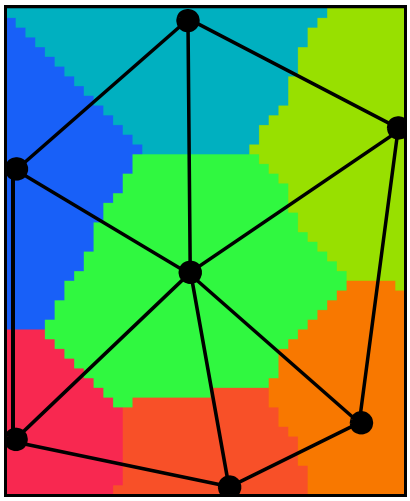


Figure 6.15 The Voronoi polygons and its dual product (i.e. the triangles).

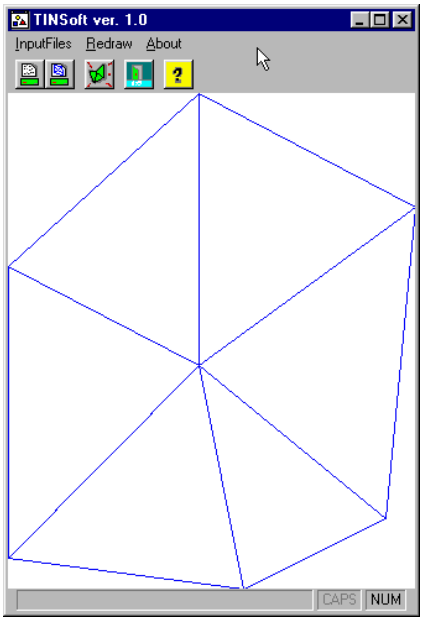


Figure 6.16 The detected TINs from the Voronoi tessellated image.

The triangle detection algorithm implementation works. Figure 6.15 and Figure 6.16 indicate the workability of the algorithm.

6.4.1 TIN topological data structuring

A program has been developed for establishing TIN neighbour information (i.e. TIN topology). With this one could determine the neighbours (neighbouring triangles) of given triangles. This is very useful for some applications using the TIN data structure. The algorithm to establish the neighbour triangles is based on the following concept: a triangle neighbour is found if two common nodes of the triangles are encountered. One triangle may have a maximum of three different neighbours. Below is the pseudo-code for the algorithm, where == means logical equality, ++ means increment and && means logical AND:

```
loop (from triangle(t) = 1 to last)
    {
     loop (from triangle(tt) = 1 to last && num of neighbour <= 3)
      {
       if (t == tt) continue;
```

```
set CommonNode = 0;

        loop (from node = 0 to < 3) && (CommonNode <= 2)
          {
           CheckNode =   (tri[t] -> Node[i] == tri[tt] -> Node[0]) ||
                         (tri[t] -> Node[i] == tri[tt] -> Node[1]) ||
                         (tri[t] -> Node[i] == tri[tt] -> Node[2]);

           if (CheckNode == true)
            {
             CommonNode ++;              // increase the common node
             if (CommonNode == 2)
              {
               NumofNbr ++;              // increase the number of neighbour
               Nbr[NumofNbr] = tt;       // this triangle
               TotalNeighbour = NumofNbr + 1;// set total neighbours for a triangle
              }
            }
          }
        }
      }
```

The input is a TIN file (an ascii file of three triangle nodes; Node1, Node2, Node3), and the output is an NBR file (a file of triangle number, number of neighbour, Neighbour[1] or (Nbr1), Neighbour[2] or (Nbr2), and Neighbour[3] or (Nbr3)), see Figure 6.17. The links of the dotted circles show that the triangle T1 and the triangle T2 are neighbouring triangles.

.

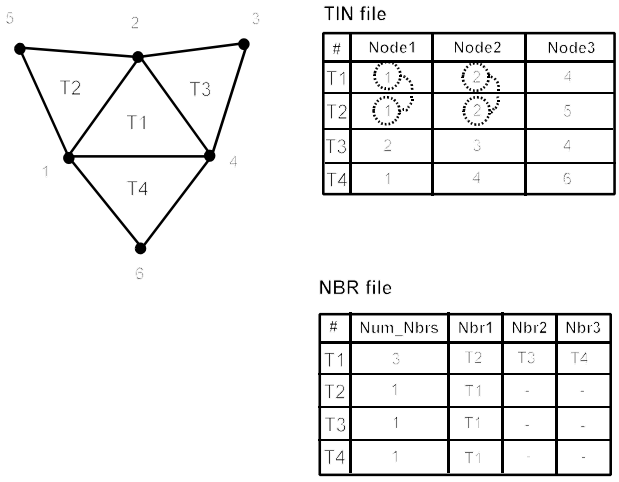


Figure 6.17 The TIN neighbour data structure

Full neighbouring information for the triangles is well described in the NBR file, and the link of the XYZ coordinates with the TIN file (Figure 6.18) facilitates other tasks such as visualization.

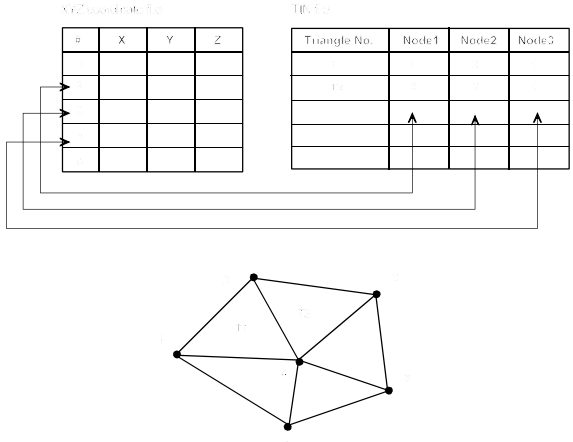


Figure 6.18 The link of XYZ coordinates and the TINs

6.5 Visualization

It has been claimed by de Berg (1997) that the visualization of TINs is one of the major issues in TIN development. In this work only a simple display program for visualizing the generated TINs has been developed. One of the fundamental tasks of any GIS or DTM package is to visualize data. Figure 6.20 shows a simple TIN visualization.

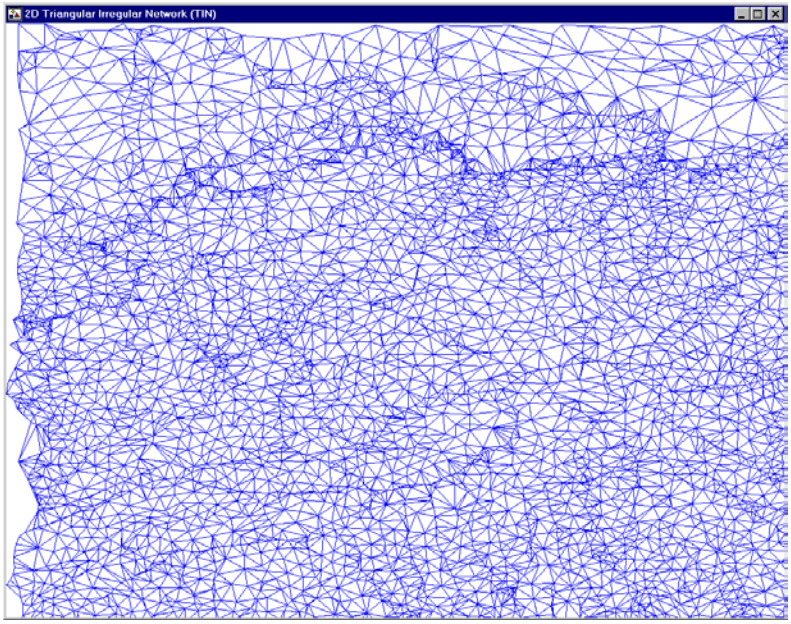


Figure 6.19 The visualization of TINs generated using digitized contours data sets.

The visualization program takes two input files, a XYZ coordinate file, and the TIN table file. The triangles three nodes (i.e. Node1, Node2, and Node3) can be linked to the corresponding XYZ coordinate table for the nodes with the appropriate pointers. Based on values in the XYZ file, triangles could be shaded according to slope, elevation etc., for further visualization

6.6 3D Distance Transformation

Digital distance transformations in 3D have been considered for more than a decade not only in medical imaging but also in other areas (Borgefors, 1996). In this work, the DT technique was used to generate a DT image, a Voronoi image and tetrahedra. The 2D DT algorithm discussed in previous sections can be extended to the third dimension relatively straightforwardly due to the nature of the raster data structure. Thus, the

same DT principle is utilised for the 3D TIN development. A 3D mask of dimension 3 x 3 x 3 was used as proposed by Borgefors (1996) known as Chamfer 3-4-5 mask, see Figure 6.20. Other types of masks are also applicable such as the Chessboard mask and the City-block mask (Borgefors, 1996).

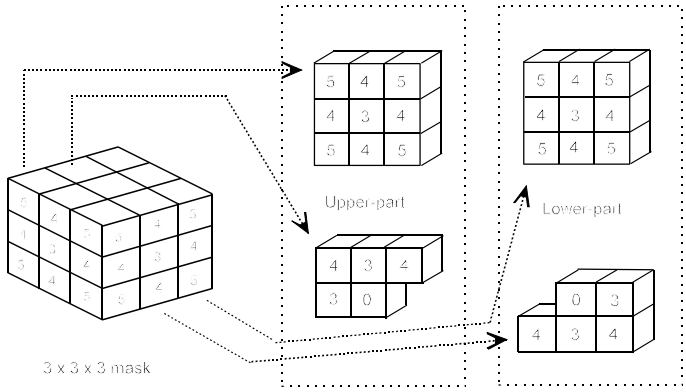


Figure 6.20 The 3-4-5 mask for the 3D DT

The Chamfer 3-4-5 mask is used due to its computational simplicity and its ability to generate quite accurate distance images. Each voxel in the mask is assigned a local distance either with a value 3, 4 or 5, depending on the voxel location, again see Figure 6.20. The centre voxel of the mask is surrounded by 26 other voxels in x, y, z directions, where each voxel has three types of voxel neighbours. They are called face neighbours, edge neighbours and node or vertex neighbours. The face neighbour voxels are assigned the value 3, the edge voxels the value 4, and the vertex voxels the value 5.

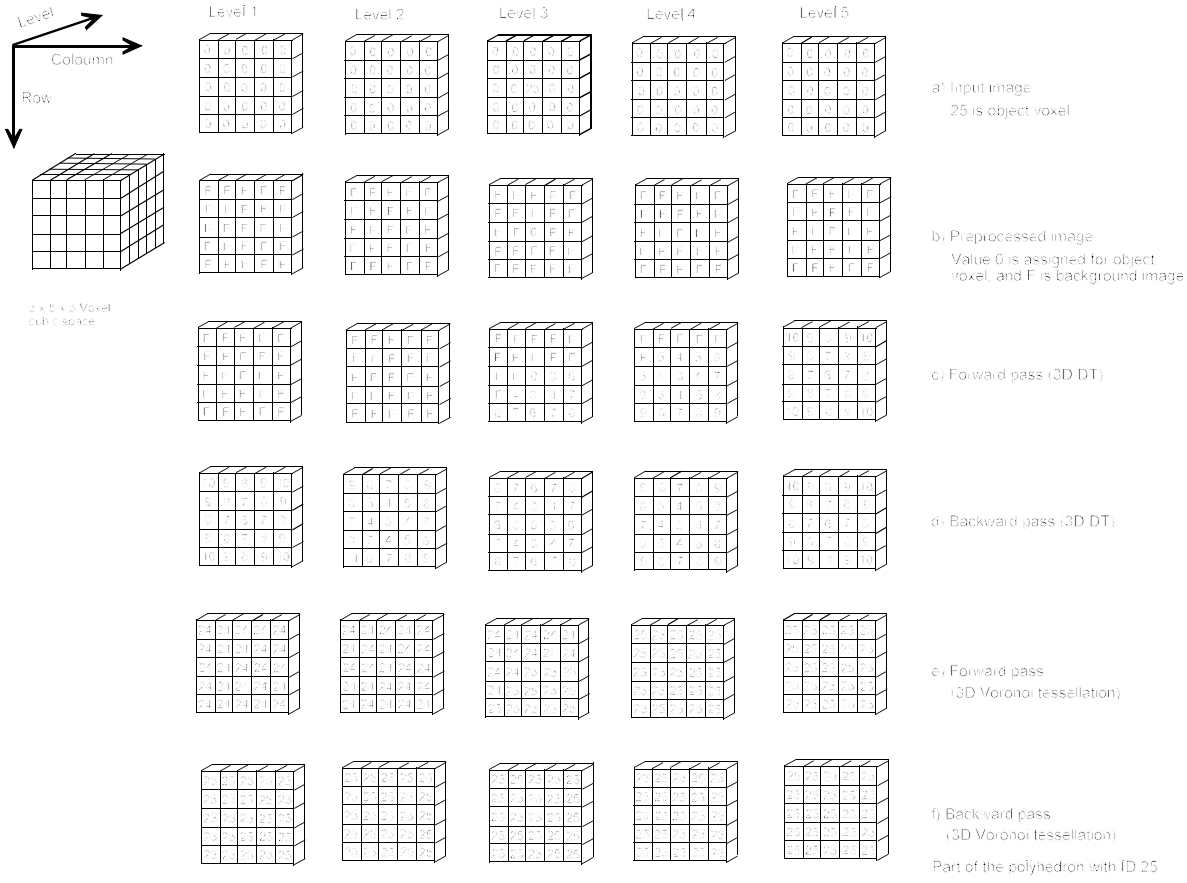


Figure 6.21 Slice of images (along the Z or level direction) for the 3D DT and 3D Voronoi tessellation

Figure 6.21 shows how the voxel values are accumulated within a (5 x 5 x 5) voxel space in the DT and Voronoi operations. To generate a distance image of a 3D raster image, the first step is to set the voxel background image to the highest integer value (F) and the object voxels to zeros (i.e. 0), see b. The image is then scanned in two passes, i.e. forward and backward passes. The forward pass (using the upper-part mask) begins from the first voxel to the last voxel. At this stage, the voxels surrounding the object voxels will get new values. The new value is the minimum distance from the 14 possible voxel candidates (see c). The result of the first pass is taken into account for the second pass. This time, the image is scanned with the lower-part mask (i.e. the backward pass) beginning from the last voxel and moving to the first voxel, again see Figure 6.21 for the

accumulated distance of a 5 x 5 x 5 cubic space (see d). A 3D distance-transformed image is formed after the two passes are carried out (see Figure 6.22). The Figure 6.22 shows the graphic output of the 3D DT of several random points in 3D space.

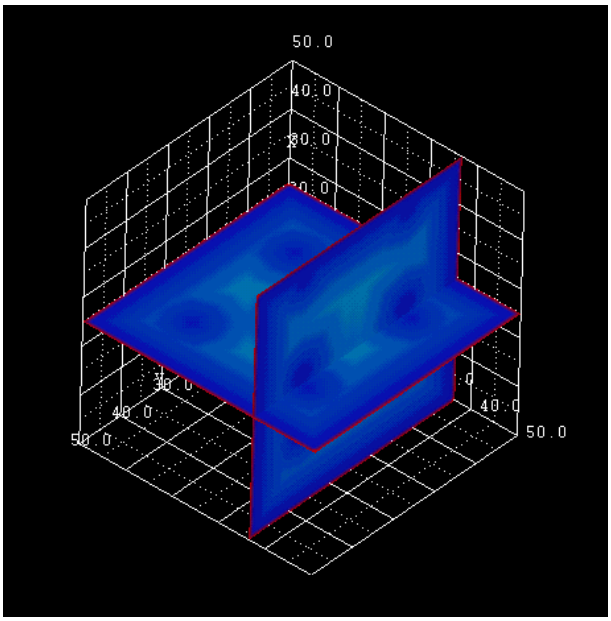


Figure 6.22 An example of a 3D distance transformation image of four points shown as double cross-sections of a 3D space (visualized via the AVS software) in the x, y and x, z planes.

The algorithm for the 3D DT in a pseudo-code follows:

```
// Procedure to set the background image
void set background()
 {
  loop from first row to last row
   {
    loop from first column to last column
     {
          Set loop for (first level to last level
          {
          if (Pixel value not equal to background)

          Set Pixel value to zero;
          else
          Set Pixel value to background (highest possible value);
          }
```

```
      }
    }
  }

// Procedure to assign the Upper Mask
void GetUpperMask()
  {
   Assign the Mask[0] to Mask[4] to their corresponding pixel locations,
        e.g.  Mask[0] = Pixel[row-1][column-1];

// Procedure to assign the Lower Mask
void GetLowerMask()
  {
   Assign the Mask[4] to Mask[8] to their corresponding pixel locations,
        e.g.  Mask[4] = Pixel[row][column];
  }

// Procedure to compute distance in forward pass
void ForwardPass()
  {
   loop from first row to last row
    {
     loop from first col to last col
      {
       GetUpperMask();
       If Mask has odd index add 4 to the Mask value;
       else
       Add 3 to the Mask value;
        }
       Get the minimum value of Mask[0] to Mask[4] and assign to this pixel;
      }
    }
  }

// Procedure to compute distance in backward pass
void BackwardPass()
  {
   loop from last row to first row
    {
     loop from last column to first column
      {
       GetLowerMask();

      If Mask has odd index add 4 to the Mask value;
       else
       Add 3 to the Mask value;
        }
       Get the minimum value of Mask[0] to Mask[4] and assign to this pixel;
```

```
    }
   }
}
```

and finally we need to combine the above steps into the following step:

```
// Procedure to compute the distance using forward and backward
void Forward&Backward()
 {
  Reads the input Image;
  Set the Background;
  Compute distance using the ForwardPass;
  Compute distance using the BackwardPass;
  Write and save the transformed image to file;
 }
```

6.7 3D Voronoi Tessellation

A Voronoi image is generated from the DT image. Again, these two images are generated in parallel. The task also involves three steps. First, cover the image with the mask. Second, the values of the mask are added to the value of the voxels being covered by the mask. Third, a minimum value from the 14 voxel candidates is determined and assigned to the current voxel location. The original voxel value of the current voxel location is taken, assigned, and written to the 3D Voronoi file. This is done prior to the mask being moved to the next voxel location. The process continues until the last voxel of the image is reached. Again, the result of this forward pass is taken into account in the backward pass which begins from the last voxel and proceeds to the first voxel of the image. Figure 6.21 (e and f) shows how the 3D Voronoi polygons (i.e. polyhedrons) were generated from one object voxel with ID = 25. In other words, a polyhedron of the voxels with ID 25 was created. Visualization of the 3D DT and 3D Voronoi images or polyhedrons can be achieved by a true 3D viewing package as provided by the AVS™ software, see Figure 6.23. Basic operations for visualizing these 3D images via the AVS is provided in the Appendix E.

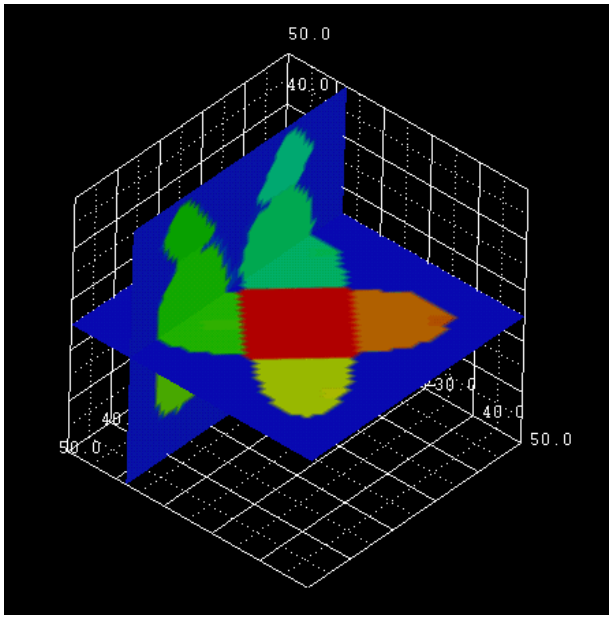


Figure 6.23 An example of 3D Voronoi tessellation of four points shown as double cross-section of 3D space (visualized via the AVS software)

The algorithm for the above 3D Voronoi tessellation in pseudo-code can be written as:

```
// Procedure: SetBackground
void SetBackground(Voxel3D Voxel, unsigned char Bg, unsigned char Fg)
 {
 loop from first level to last level
   {
    loop from first row to last row
     {
      loop from first column to last column
       {
        if (Voxel[l][row][col] == 0)
           Voxel[l][row][col] = Bg;
        else
        if (Fg > 0)
           Voxel[l][row][col] = Fg;
        }
     }
   }
 }

// Procedure: GetUpperMaskDist
void GetUpperMaskDist(int l, int r, int c, Voxel3D Voxel, Mask& MaskPix)
 {
  Assign the MaskPix[0] to MaskPix[13] to their corresponding Voxel locations.
```

```
        e.g. MaskPix[0] = Voxel[l-1][r-1][c-1];
 }

// Procedure: GetLowerMaskDist
void GetLowerMaskDist(int l, int r, int c, Voxel3D Voxel, Mask& MaskPix)
 {
  Assign the MaskPix[13] to MaskPix[26] to their corresponding Voxel locations.
        e.g. MaskPix[13] = Voxel[l][r][c];
 }

// Procedure: GetUpperMaskVoronoi
void GetUpperMaskVoronoi(int l, int r, int c, Voxel3D VoxelVor, Mask& MaskPixVor)
 {
  Assign the MaskPixVor[0] to MaskPixVox[13] to their corresponding Voxel locations,
        e.g. MaskPixVox[0] = VoxelVor[l-1][r-1][c];
 }

// Procedure: GetLowerMaskVoronoi
void GetLowerMaskVoronoi(int l, int r, int c, Voxel3D VoxelVor, Mask& MaskPixVor)
 {
  Assign the MaskPixVor[13] to MaskPixVox[26] to their corresponding Voxel locations,
        e.g. MaskPixVox[13] = VoxelVor[l][r][c];
 }


// Procedure: ForwardPass
void ForwardPass(Voxel3D Voxel, Voxel3D VoxelVor)
 {
 loop from first level to last level
   {
    loop from first row to last row
     {
      loop from first column to last column
       {
        GetUpperMaskDist(l, r, c, Voxel, MaskPix);
        GetUpperMaskVoronoi(l, r, c, VoxelVor, MaskPixVor);
        for (k = 0; k < 13; k ++)
          {
           if ((k == 0) || (k == 2) ||
               (k == 6) || (k == 8))
             MaskPix[k] = MaskPix[k] + 5;

           if ((k == 1) || (k == 3) ||
               (k == 5) || (k == 7) ||
               (k == 9) || (k == 11))
             MaskPix[k] = MaskPix[k] + 4;

           if ((k == 4) || (k == 10) || (k == 12))
```

```
            MaskPix[k] = MaskPix[k] + 3;
          }

        if (MaskPix[13] != 255)
          MaskPix[13] = 0;

        Voxel[l][r][c] = MaskPix[MinByIndex(0, 13)];
        VoxelVor[l][r][c] = MaskPixVor[MinByIndex(0, 13)];
       }
     }
   }
 }


// Procedure: BackwardPass
void BackwardPass(Voxel3D Voxel, Voxel3D VoxelVor)
 {
  loop from last level to first level
   {
    loop from last row to first row
     {
      loop from last column to first column
       {
        GetLowerMaskDist(l, r, c, Voxel, MaskPix);
        GetLowerMaskVoronoi(l, r, c, VoxelVor, MaskPixVor);
        for (k = 26; k > 13; k --)
         {
          if ((k == 18) || (k == 20) ||
              (k == 24) || (k == 26))
            MaskPix[k] = MaskPix[k] + 5;

          if ((k == 19) || (k == 21) ||
              (k == 23) || (k == 25) ||
              (k == 15) || (k == 17))
            MaskPix[k] = MaskPix[k] + 4;
          if ((k == 14) || (k == 16) || (k == 22))
            MaskPix[k] = MaskPix[k] + 3;
         }
        Voxel[l][r][c] = MaskPix[MinByIndex(13, 26)];
        VoxelVor[l][r][c] = MaskPixVor[MinByIndex(13, 26)];
       }
     }
   }
 }


// Procedure: ForwardVoronoi
```

```
void ForwardVoronoi()
 {
  ReadVoxelImage(Voxel);
  CopyVoxel(Voxel, VoxelVor);
  SetBackground(Voxel, 255, 0);
  ForwardPass(Voxel, VoxelVor);
 }


// Procedure: BackwardVoronoi
void BackwardVoronoi()
 {
  BackwardPass(Voxel, VoxelVor);
 }
```

6.8 Tetrahedron Network (TEN) Generation

Using the same principle as for the 2D TIN, the algorithm for the 3D TIN utilised a mask of 2 x 2 x 2, see Figure 6.24. It has 8 voxel elements. It provides a unique way of establishing tetrahedra. In order to obtain non-overlapping tetrahedra, several predefined conditions have to be imposed during voxel scanning. There are 6 possible non-overlapping tetrahedra that we can get from the mask shown in Figure 6.24.

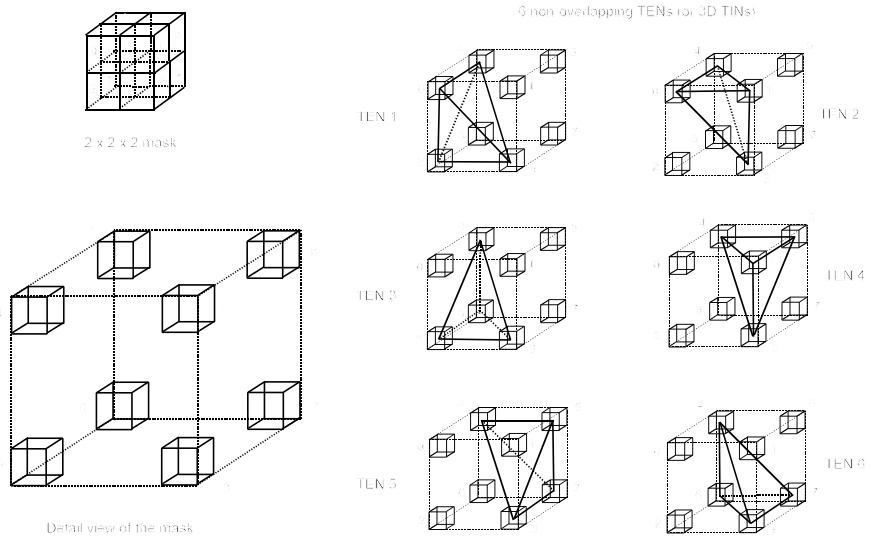


Figure 6.24 The six non-overlapping TENs

The mask is then used to scan the voxel's Voronoi tessellated-image once. In order to detect non-overlapping TEN, several conditions has to be imposed during the voxel scanning. The imposed conditions follows:

For TEN 1: $0 \neq 2 \neq 3 \neq 4$, For TEN 2: $0 \neq 1 \neq 3 \neq 4$

For TEN 3: $2 \neq 3 \neq 4 \neq 6$, For TEN 4: $1 \neq 3 \neq 4 \neq 5$

For TEN 5: $3 \neq 4 \neq 5 \neq 7$, For TEN 6: $3 \neq 4 \neq 6 \neq 7$

Once a tetrahedron was detected (based on the imposed conditions), it is then written to a file. The file contains a record of tetrahedra where each record has 4 nodes, it is an ASCII file and structured as in Figure 6.25. Thus, it is one way of establishing a simple tetrahedral data structure. The data structure together with a table of nodes' coordinates provide a means for further manipulation of the data, e.g. visualization.

The algorithm was implemented and tested by using simulated 3D raster data sets. This data set was generated by the 3D point-to-raster program developed in this work. A wireframe display program was also developed for visualizing the TENs, see Figure 6.26 for the output display.

Points table

#	X	Y	Z
0	X	Y	Z
1	X	Y	Z
2	X	Y	Z
3	X	Y	Z
4	X	Y	Z

TENs table

#	Node1	Node2	Node3	Node4
1	0	2	3	4
2	0	1	3	4
3	2	3	4	6
4	1	3	4	5
5	3	4	5	7

Figure 6.25 TEN data structure (for TEN 1 as shown in Figure 6.24).

```
// The main program has the following routines
void main()
 {
  GetVPIfile(VPI);
  GetVPDfile();
  AllocateMemory();
  Get3DTINfile();
```

```
Make3DTIN();
DeallocateMemory();
}
```

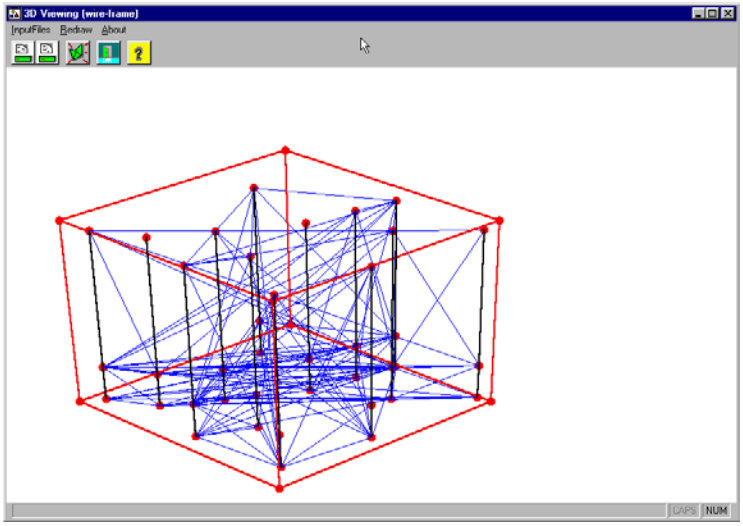


Figure 6.26 An example of TEN visualization

6.9 Constrained Triangulations

Constrained triangulation development is meant to accommodate linear features, e.g. terrain breaklines, drainage lines, faults and other linear features such as roads, railways, etc. Previously, this triangulation only worked with points as discussed in section 6.4, but now a new feature is introduced in this work, that is the capability of handling linear features. In a constrained triangulation, these lines or linear features become part of the triangle edges. Since the triangulation in this work is based on raster data, then also further development of the rasterization routine has been necessary so that it can accommodate straight lines.

The next section discusses the line rasterization algorithm and the results of constrained triangulation.

6.9.1 The line rasterization

Line rasterization is used to rasterise a series of lines - as input to the constrained triangulation. In this, a line has a start node and an end node. Thus, the rasterization is simply a process of calculating the position of the pixels between these two end nodes of a line. Line rasterization was successfully implemented for the constrained triangulation by incorporating several concepts, including the concept of the line equation, y = mx + C (where *m* is a slope of a line), the concepts of the Bresenham line plotting algorithm and the concept of the Tang (1992) algorithm. The *m* (slope of a line) is used only to detect the type of a line or arc in the data file rather than to do the rasterization of points of a line. A line or an arc could have a slope of (0 < m < 1), or (m > 1), or (-1 < m < 0), or (m < -1) or is vertical or it could be horizontal. This approach can handle all cases of lines, i.e. in all quadrants. The Bresenham algorithm is used to speed up the operation (see below), and Tang's algorithm is for the constrained triangulation where the edge or arcs can be accommodated in the Voronoi and the triangulation process. This approach of line rasterization is fast and produces well rasterised lines. It is fast because it does not involve a divisional operation during the pixel increment along the line. The Bresenham line algorithm can be found described in computer graphics texts such as Foley *et al* (1996), and Farrell (1994). Below is the line rasterization pseudo-code (only the first quadrant is presented). (The rest of the quadrants are attached in the Appendix J (Volume II)).

```
void LineRasterization()
    {
    GetXYZFile();
    GetArcFile();
    loop from first arc to last arc
        {
        GetXYZforArcNodes(t, xstart, ystart, xend, yend, sNd, eNd);
      Calculate the m for each line (m = (yend - ystart) / (xend - xstart));
        GetMiddleXY for each line;
        Detect the slope of a line, it could be (m > 0) && (m < 1), or
              (m > 1), or (m < 0) && (m > -1), or (m < -1), or


              vertical, or horizontal line;
```

```
if (m > 0) && (m < 1) do the following
    {
    Swap the coodinates of the two nodes so that the operation begins
        with the lower position node;
    Calculate the dx, dy;
    Calculate the (2 * dx) and (2 * dy);
    Calculate the pixel location, i.e. the row, and col of the pixel location;
    if swap the nodes is true
        Assign the pixel location with the correct value, i.e. the correct node number;
    else
        Assign the pixel location with the correct value, i.e. the correct node number;
    Initialise the Bresenham error of a line (that is the difference of a point to the true position, see
    Bresenham algorithm for detail.
    Increase the xstart (i.e., xstart ++)
    while (xstart < xend)
        {
            If the error > 0
                error = error + (2*dy - 2*dx);
                ystart ++;
            else
                error = error + (2*dy);

            Calculate the row and col of the pixel location;
            Assign the correct pixel value to this location;
            Increase the xstart (i.e., xstart ++);
        }
and do the rest of the line cases (i.e. for the other quadrants).
}
```

This line rasterization algorithm produces the following result as tested with several simulated nodes, and arcs, as shown in Figure 6.27.

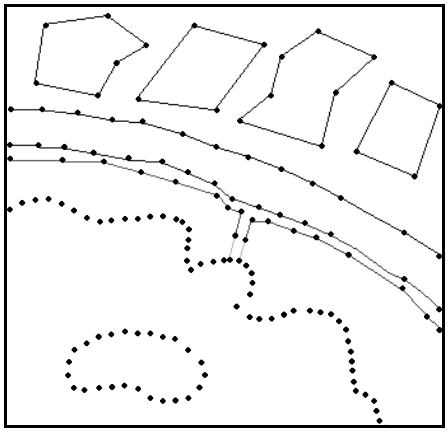


Figure 6.27 Screen shot of the rasterised nodes and arcs (Note: the nodes purposely made larger to show the location of the nodes).

6.9.2 The construction of the constrained TINs

In order to generate the constrained triangulation, rasterised points and rasterised lines are needed as an input. This constrained triangulation is based on the concept presented by Tang (1992). Here, the constrained edges were represented by a series of pixels whose values were based on the edge node IDs (identifiers). In other words, half of a line was represented by the pixels of the start node, and the other half by the pixel values of the end node, see Figure 6.28.

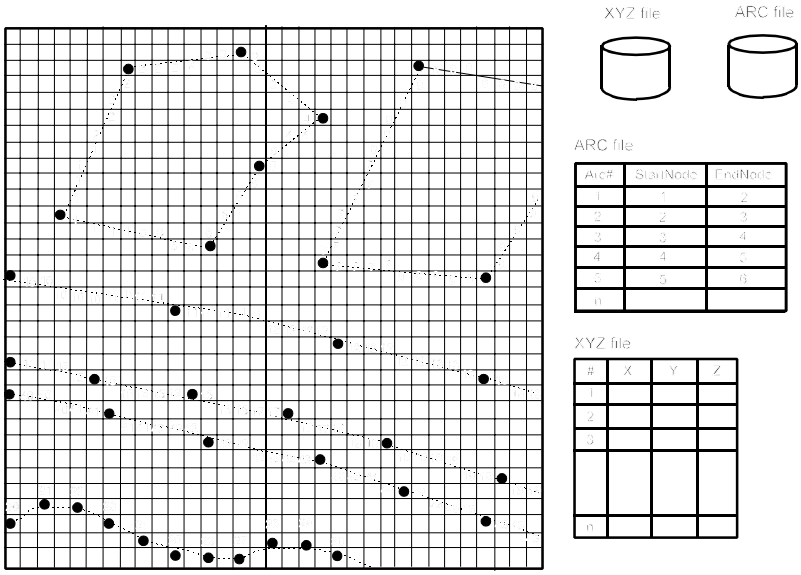


Figure 6.28 An example of the pixel locations of the rasterised points and several edges or arcs. The left side is the corresponding coordinates and arc files.

The above figure also shows the propagation of the pixel values of the start node to the end node of each arc. This approach also conforms with the Voronoi tessellation concept where two kernel points have two corresponding Voronoi zones separated by a boundary which happens to be located in the middle of the two kernel points.

The illustrations Figure 6.29a to Figure 6.29d show the result of the DT and the Voronoi tessellation implemented for the constrained edges.

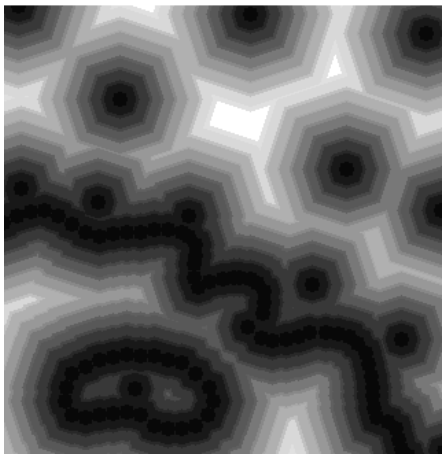


Figure 6.29a The DT image of the rasterised kernel points

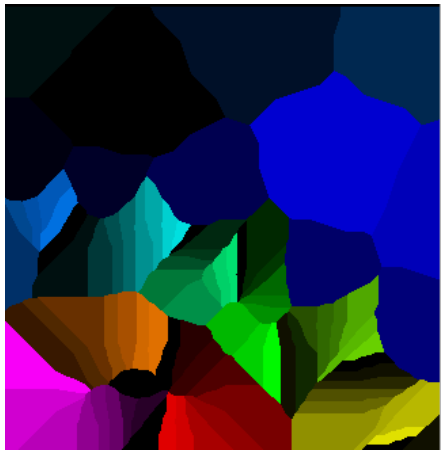


Figure 6.29b The Voronoi image for the kernel points of Figure 6.29a

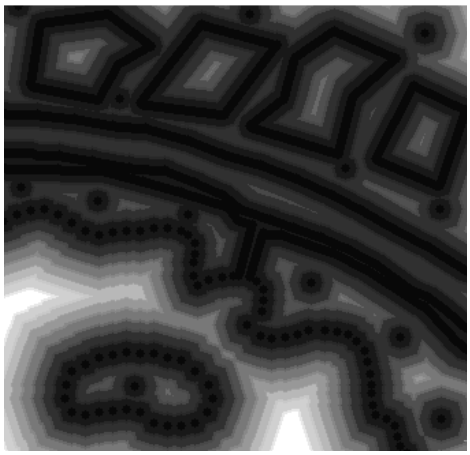


Figure 6.29c The DT image with the edges and points

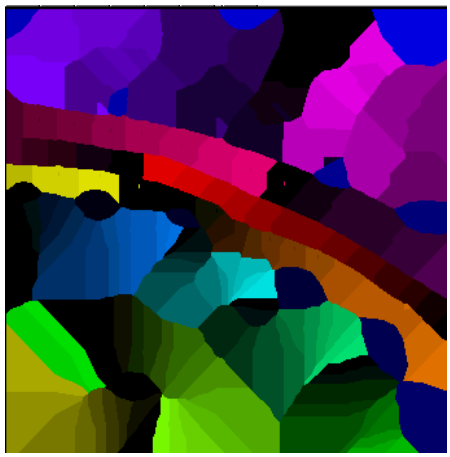


Figure 6.29d The Voronoi image of the corresponding edges and points of Figure 6.29c

It is clearly shown that the edges or arcs can be accommodated as a constrained feature in the distance transformation and Voronoi tessellation. A constrained edge is represented by the thick black lines as in Figure 6.29c and the respective polygons are shown in Figure 6.29d.

Further, all the points and the edges are then triangulated, and the results are the constrained triangulations, see Figure 6.31.

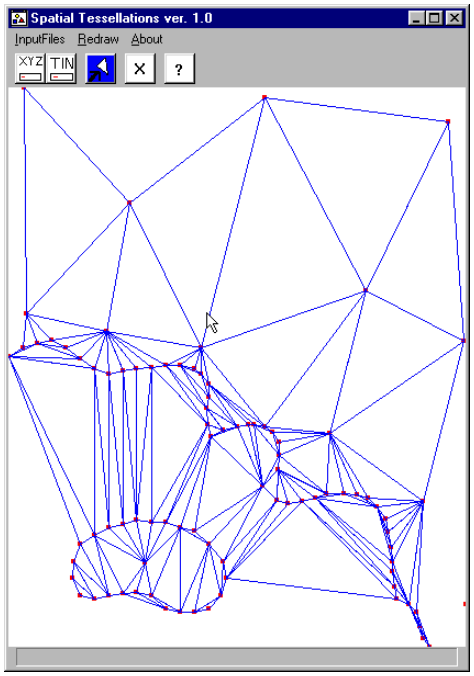


Figure 6.30 The generated unconstrained triangulation

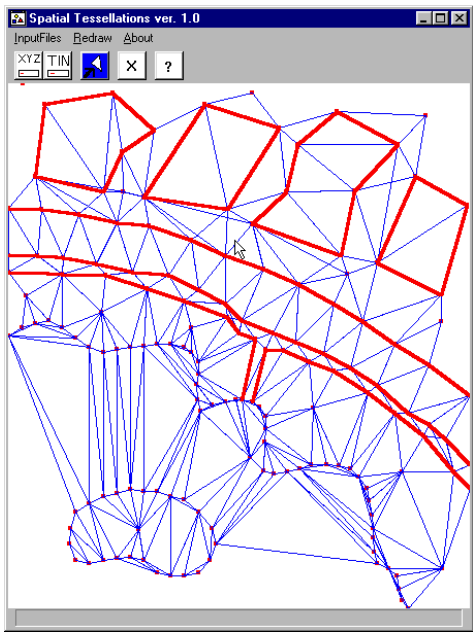


Figure 6.31 The generated constrained triangulation

In this particular example, the edges are part of the features on the terrain and also form part of the triangle edges. The results indicate that the constrained triangulation works. The development provides useful data structuring mechanisms for TIN-based spatial data modelling and the related applications. The fundamental GIS data types, i.e. node, arc, surface and volume are generated with this approach. Their related spatial modelling were discussed in Chapter 5.

The technique was also tested using photogrammetrically acquired data (Drumbuie,

Kyle of Lochalsh, north-west Scotland) - see the results in the following Figures 6.32 to Figure 6.35.

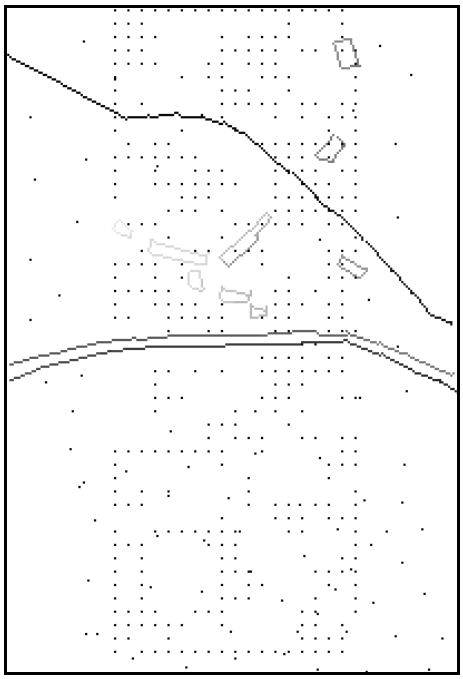


Figure 6.32 The rasterised points and lines.

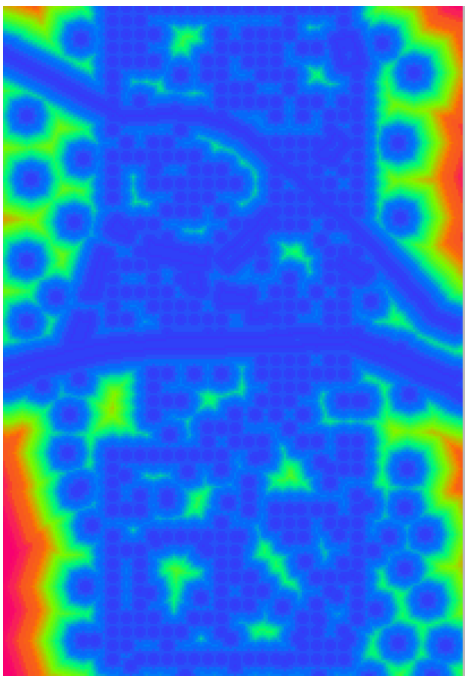


Figure 6.33 The DT image of the area.

Figure 6.34 The Voronoi image

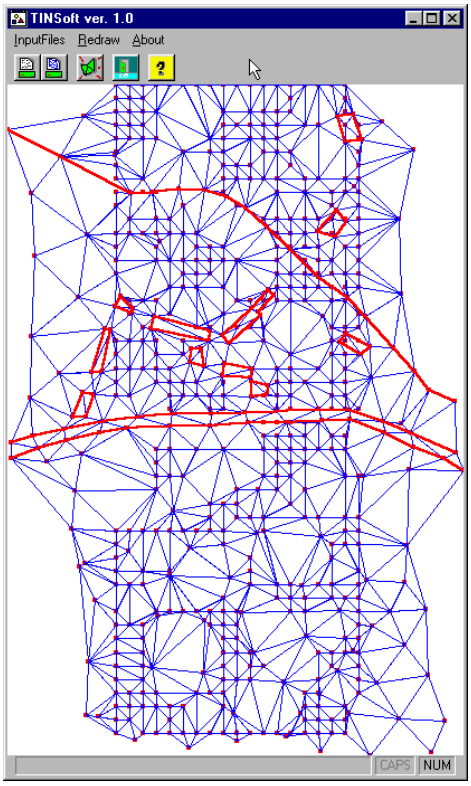


Figure 6.35 The generated TINs

6.10 Contouring Algorithm

Contouring is one of the GIS applications that has been developed for this work. This section describes the development of the data structures for the contouring and the contouring algorithm. A suitable file format for the above application is also developed, so that it could be imported to other commercial GIS software, e.g. Arc/Info, and ILWIS.

6.10.1 Data structures for contouring

This is one of the important components where the data storage and data's accessibility influence the behaviour and performance of the software. In this it should be noted that, besides the two TINs structures, namely, the three-nodes table (TRI) and the triangle neighbour (NBR), two other data structures were developed. These two TIN topological structures are a triangles' edge and right and left triangles (TRS), and a triangles' three sides (SID). The file format of these structures is shown in the Appendix D.

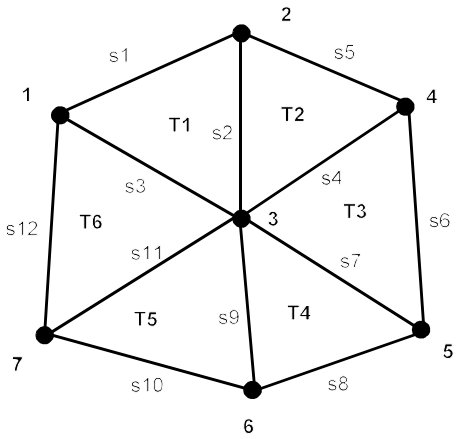


Figure 6.36 An example of 6 TINs with 7 nodes, and 12 sides or edges

Figure 6.36 shows a simple configuration of 6 TINs with 7 nodes, and 12 triangle sides.

To facilitate the contouring application, a program is developed to generate two more data structures. Thus, from TRI (Triangle # and 3 nodes) and NBR (Triangle neighbours), TRS and SID structures are generated. The TRS contains Triangle sides and Right-Left triangles while the SID structure contains Triangle # and the 3 sides or edges (see the Appendix D for the file formats).

The algorithm for converting the TRI and NBR structures to TRS and SID structures is based on the following concept: a triangle side has two nodes, and each side only has either a right triangle or a left triangle, see Figure 6.36. The software has the following routines:

```
Read the triangles
Read the triangles neighbours, and
MakeTRSandSID structure.
```

The MakeTRSandSID has the following sub procedures, the ExistingSide and the DoSide functions, and the algorithms are described below in pseudo-code, where :: means scope resolution operator between class name and methods in the class:

```
ConvertStruct :: bool ExistingSides(int n1, int n2, int& s)
        {
        do
          {
           found = (n1 == Node1) && (n2 == Node2);
           if (! found)
             s ++;
          } while ((! found) && (s <= nsid));

         return found;
        }

       void ConvertStruct :: DoSide(int t, int n1, int n2, int snbr)
        {
        if (n1 > n2)
          {
           h = n1;
```

```
   n1 = n2;
   n2 = h;
  }

 if (ExistingSides(n1, n2, s))
  {
   TriSides[s]->RightTri = t;
   Tri3Sides[t]->Side[snbr] = s;
  }

 else
  {
   nsid ++;
   TriSides[nsid] = new tsid;
   TriSides[nsid]->Node1 = n1;
   TriSides[nsid]->Node2 = n2;
   TriSides[nsid]->LeftTri = t;
   TriSides[nsid]->RightTri = 0;

   Tri3Sides[t]->Side[snbr] = nsid;
  }
}
```

TRS structure

#	Node1	Node2	RightTri	LeftTri
s1	1	2	T1	0
s2	2	3	T1	T2
s3	1	3	T6	T1
s4	3	4	T3	T2
s5	2	4	T2	0
s6	4	5	T3	0
s7	3	5	T4	T3
s8	5	6	0	T4
s9	3	6	T9	T4
s10	6	7	T5	0

SID structure

#	Side1	Side2	Side3
T1	s1	s2	s3
T2	s2	s4	s5
T3	s4	s6	s7
T4	s7	s8	s9
T5	s9	s10	s11
T6	s3	s11	s12

Figure 6.37 The TRS and SID structure

6.10.2 The algorithm

The contouring program makes use of two TIN data structures, namely the TRS and SID structure plus the coordinates, and it is based on linear interpolation. The contouring program performs the following routines:

Read the input data (coordinates, TINs structure of TRS, and SID tables).
Open the output file for the interpolated data.
Get the min and max of the XYZ input coordinates, then perform
MakeContouring.

The algorithm can be described as follows. MakeContouring has several sub methods or procedures, they are CheckSide, FindFirstTri, Interpolate, FindOtherSide, FindNextTri, and GetContours. The CheckSide is to check the side of a triangle and whether or not it can be interpolated with the user requested contour heights. The FindFirstTri is used to get the first triangle which contains the requested contour height. FindOtherSide is to get the other side of a triangle, whereas the FindNextTri is to get the next triangle in the list. The Interpolate is to compute the interpolated point once a triangle's side satisfies the imposed conditions. Then all these sub procedures are combined as the GetContours function performs the subsequent major task - the contouring. Thus the GetContours behaves as follows.

```
void MakeContouring :: GetContours(int Hreq)
 {
    do
     {
      while (FindFirstTri(t, s) != 0)
       {
        ii ++;
        SegNr = ii;
        startt = t;
        starts = s;
        done = false;
        secondpart = false;

        do
         {
          Interpolate(s, SegNr, Hreq);
          FindOtherSide(t, s, nexts);
          FindNextTri(t, nexts, nextt);
          if (nextt == startt)


           {
            // found closed contours
            Interpolate(nexts, SegNr, Hreq);
```

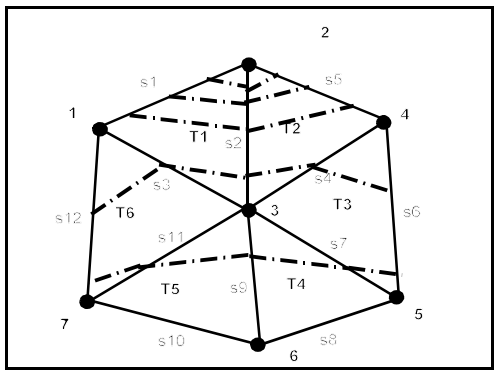


Figure 6.38 An example of contours with 35 m interval using 6 TINs with 7 nodes, and 12 sides or edges

```
                done = true;
              }
            else if (nextt == 0)
                {
                 // hit border
                 Interpolate(nexts, SegNr, Hreq);
                 if (secondpart)
                   done = true;
                 else
                   {
                    FindNextTri(startt, starts, t);
                   if (t == 0)
                       done = true;
                    else
                      {
                       s = starts;
                       secondpart = true;
                      }
                   }
                }
             else
             {
              t = nextt;
              s = nexts;
             }
           } while (! done);
        }
      } while ( ! ((t == 0) || (s == 0) || (z != prevH)) ) ;
}
```

6.10.3 The contour visualization

The contour algorithm has been tested using real terrain data sets. Figure 6.39 and Figure 6.40 illustrate the generated contours from the simulated and digitized contours datasets with different contour intervals. Format conversion programs for contours display in other popular GIS packages are also developed, for example PC Arc/Info

(.LIN format) and ILWIS packages (.SEG format). Figure 6.40 shows one of the examples of derived contours output from photogrammetrically acquired data sets.

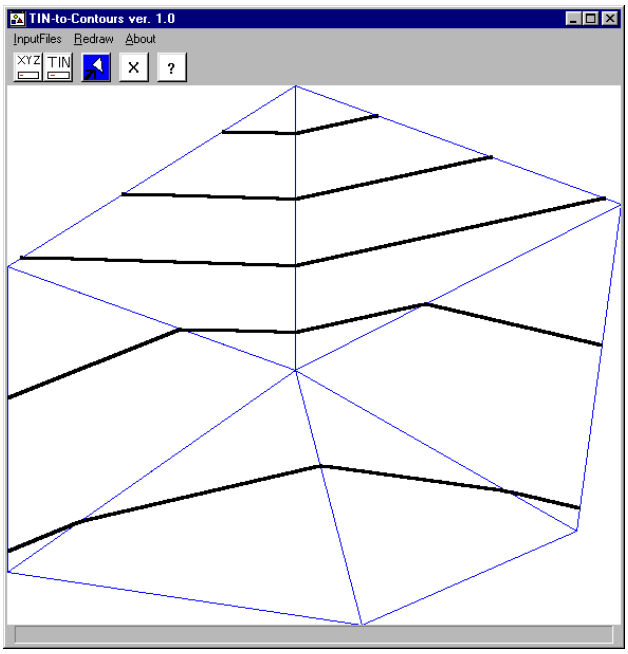


Figure 6.39 The generated contours from the simulated datasets (6 TINs)

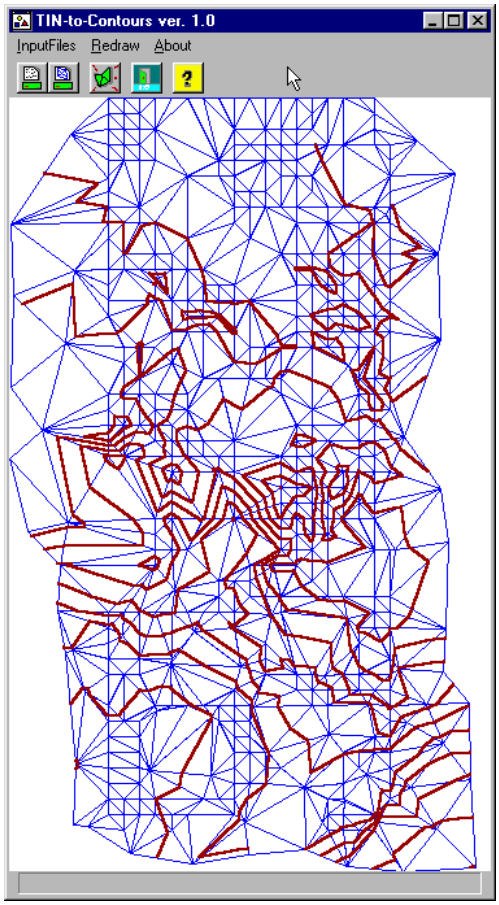


Figure 6.40 The generated contours of 4-m interval using photogrammetrically datasets (Drumbuie, Kyle of Lochalsh, north-west Scotland)

6.11 Summary

Several important algorithms for TINs (2D and 3D) constructions have been introduced, namely the DT, the Voronoi tessellations, triangulations (including constrained triangulations), constrained line rasterisation. In this thesis, constrained 3D rasterization and constrained 3D TIN have not been dealt with. This author, along with others working in the area in other institutions, have so far found the mathematical,

geocomputational, and computer science input to addressing constrained 3D TIN so challenging that little has been achieved (Zlatanova, 2000). It is envisaged that the constrained features will provide research opportunities for some time. The 3D TINbased information system will generate much useful information by having this 3D constrained capability e.g. the handling of subsurface fault lines. One of the GIS application algorithms was also introduced, that is contouring. The chapter also described other major tasks, that is the data structuring and the topological structuring for 2D TIN and 3D TIN (TEN).

The next chapter discusses the graphical user interface and represents the implementation and test of the methods developed in this work.

Chapter 7

The Implementation and Test

The algorithms which were described in chapter 6 need to be tested on real data. This is done using a user display interface, called TinSoft, which was developed in this work. It is a graphical user interface, developed to test the concepts, procedures, and algorithms in this work. It can be described in three parts:

- the interface,
- the triangulations, and
- the applications.

This chapter describes how TinSoft was constructed, and how it can be used to test the algorithms. It is mainly used for the visualization of the output from the algorithms developed in this work.

7.1 The Study Area

In this work, a real photogrammetrically acquired data set of a village and a simulated borehole data set were used for testing the algorithms and sub programs. The village (Drumbuie) is near Kyle of Lochalsh, north-west Scotland as shown in Figure 7.1 (an orthophoto image). The photogrammetric data were gathered by stereo digitising; for the building the roof line were gathered, the trees were represented by crown points and some points around a tree's perimeter, both sides of tracks and river centre lines. The borehole data is typical of many 3D data sets which become available particularly to natural scientist as they increasingly use 3D data.

Figure 7.1 The study area (orthophoto image) from which points and lines were extracted using 3D digitizing (with stereo mate)

7.2 The Interface

7.2.1 The multiple document interface

The interface of a program is to provide a means of communication between a computer system and its users. TinSoft uses the Windows platform as it provides a "standard" interface for manipulating windows, menus, icons, dialogue windows, messages, etc. For the interface construction, Borland's Object Windows Library (OWL) component available in the Borland C++ version 5.02 compiler was utilised (Borland, 1996). It is a Multiple Document Interface (MDI) environment where multiple windows of documents and other views can be displayed at once. An area where all the views are

displayed is sometime known as the client area.

A number of functions have been developed to facilitate the user display operations which include operations for input files, arranging window layouts, view redisplay, exiting the program and status bar messaging. Figure 7.2 shows the graphical interface for the developed program.

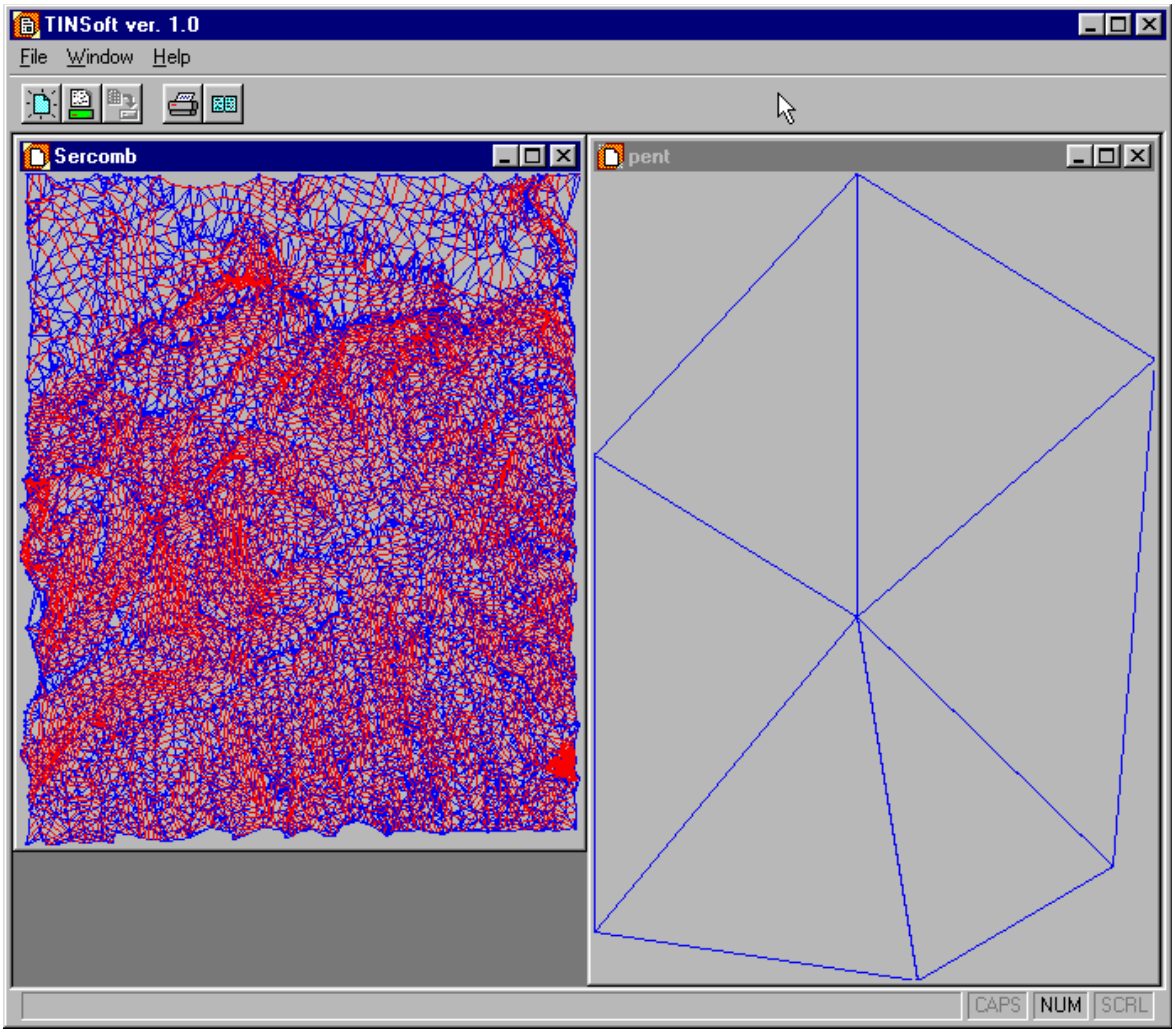


Figure 7.2 An example of the MDI of the TinSoft program. The left view shows the triangles (blue) and the derived contours (red) of the digitized datasets; the right view only illustrates the triangles using simulated data.

7.2.2 The single document interface

A single document interface (SDI) has also been developed in this work. Although this type of interface has less capability than the MDI type it has been adopted in this work for the purpose of 3D viewing for such as perspective views as well as for other display purposes as illustrated in Chapter 6. This interface has several functions such as input files and view redisplay. For ideal 3D viewing purposes, this interface needs to have an interactive viewing facility so that users could have several viewing choices; thus represents a future development recommended for this interface (see section 8.4). Figure 7.3 shows the SDI 3D viewing interface. For details of the perspective view procedure, see Appendix J (3D Viewing section) in the Volume II.

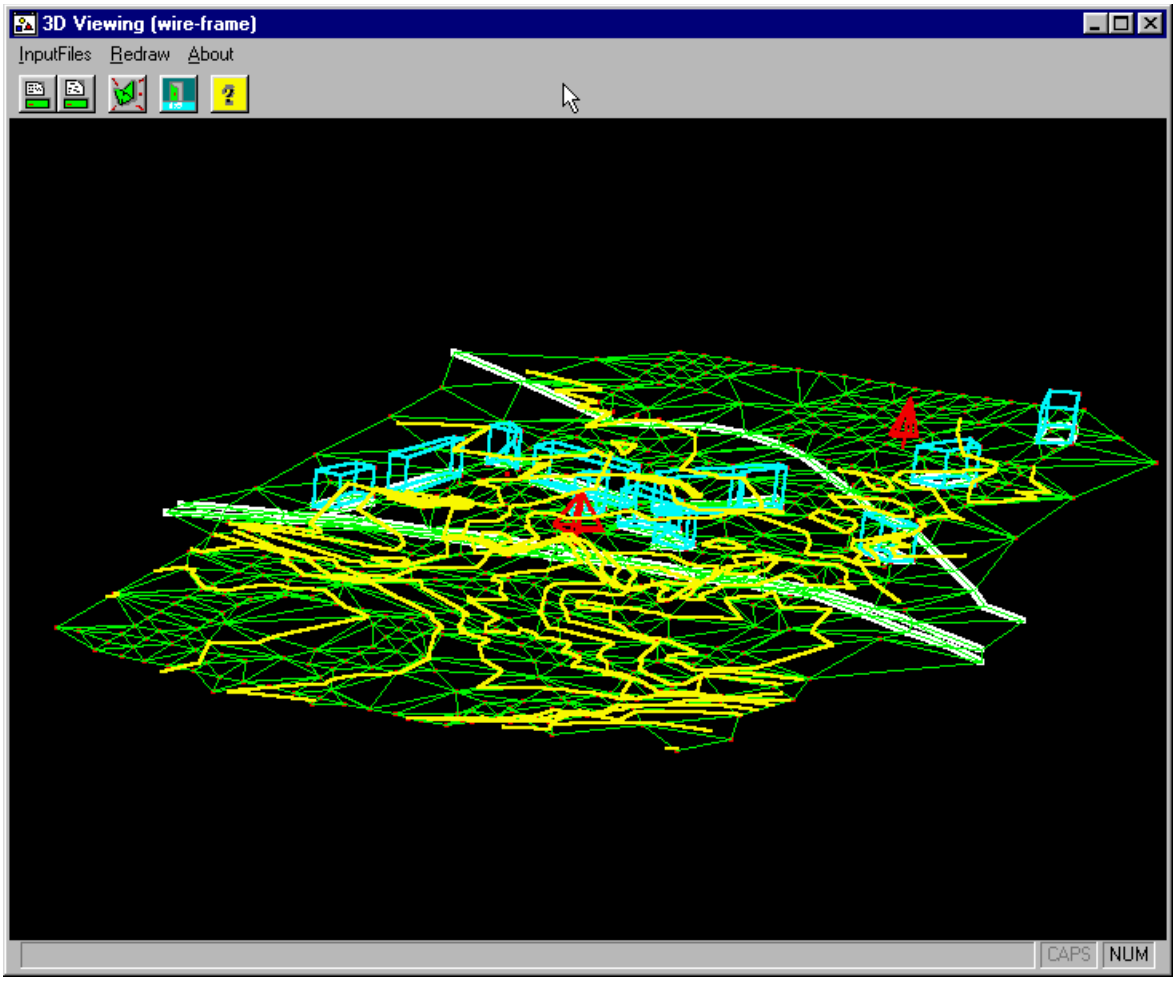


Figure 7.3 An example of the SDI interface for 3D viewing (perspective). It shows the triangles, 3D objects (buildings in blue and trees in red), linear features (roads and river centre-line with white), and derived contours (yellow) draped on top of the TIN surface.

7.3 The Triangulations: 2D and 3D TIN

Algorithms for the 2D and 3D TIN were developed as discussed in Chapter 6. Here, tests of the algorithms were made using several data sets, namely: simulated data; digitized contours; and, digital photogrammetric data. These algorithms can process several thousand terrain points. The result is shown in Figure 7.4 for 2D TIN and Figure 7.5 depicts the 3D TIN of the points along the simulated boreholes.

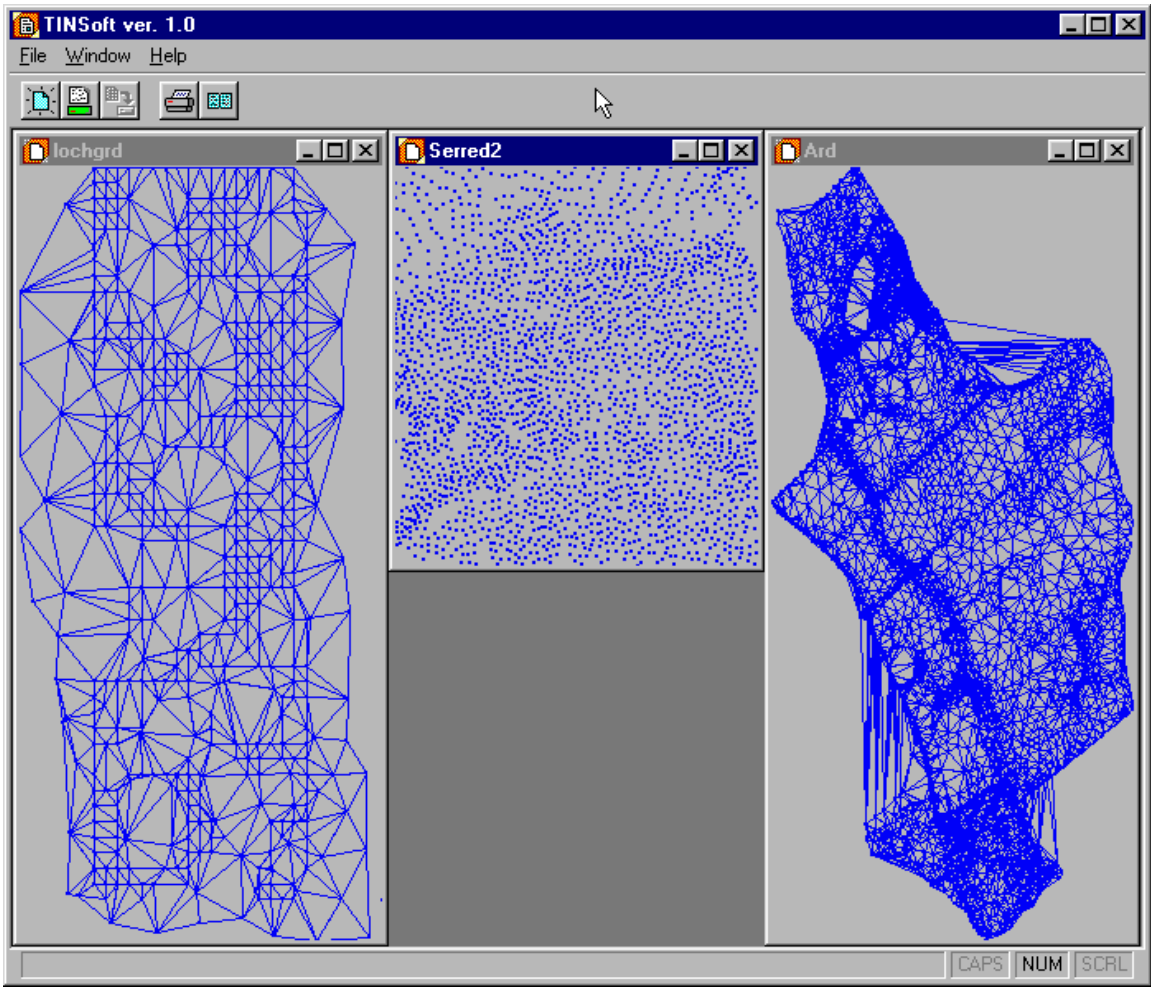


Figure 7.4 An example of the MDI interface of the 2D TIN for the three data sets (photogrammetrically derived data, digitized contours, and field survey).

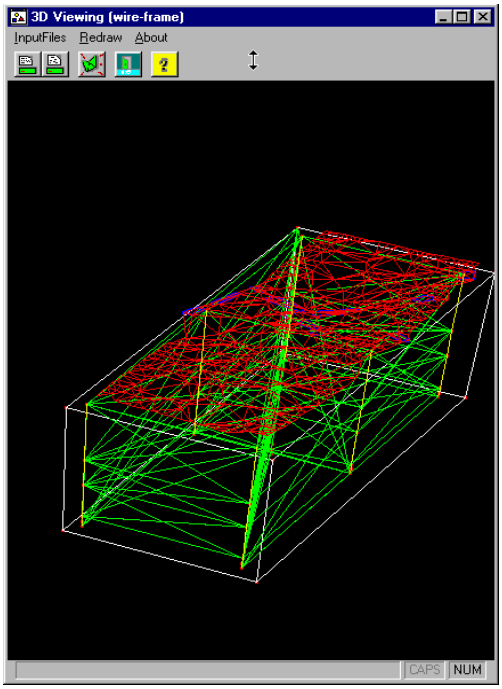


Figure 7.5a An example of the SDI interface of the simulated boreholes

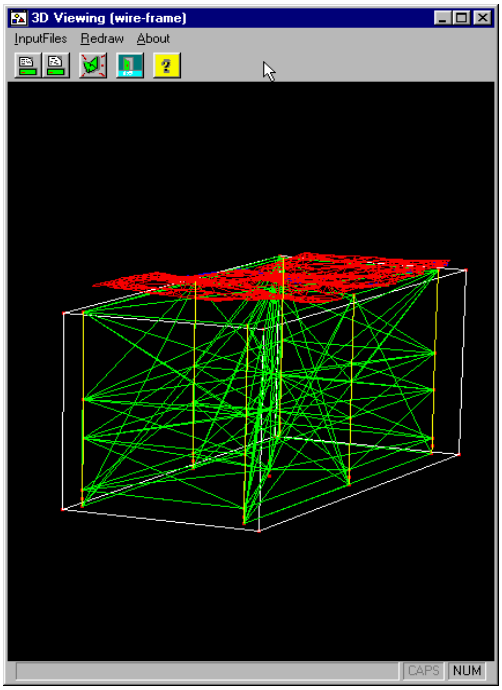


Figure 7.5b Another example of the SDI interface of the simulated boreholes with different direction of perspective view.

7.4 The Applications

7.4.1 Contouring

The contouring algorithm is tested using different contour intervals. The result of these tests is graphically depicted in Figure 7.6a, b, and c.

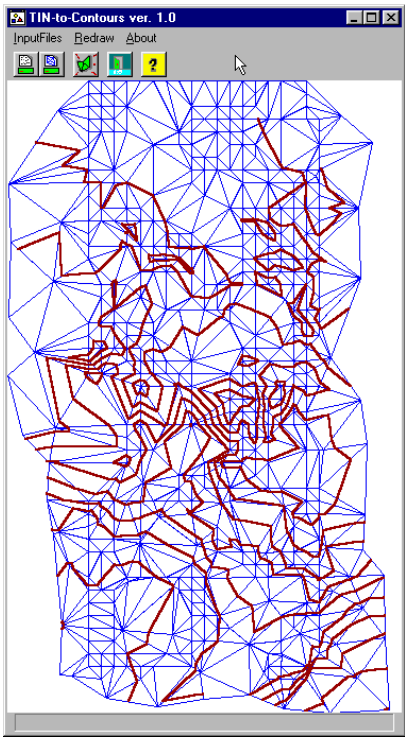


Figure 7.6a Contours of 4-m interval

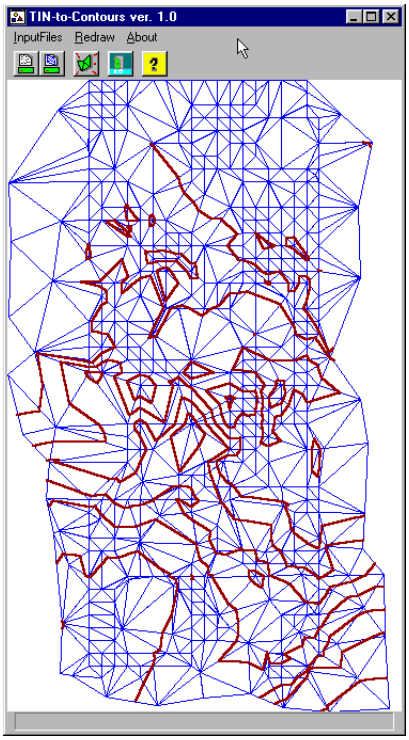


Figure 7.6b Contours of 5-m interval

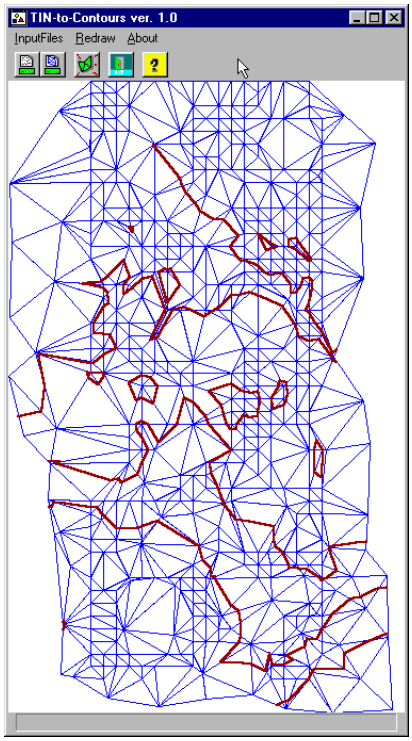


Figure 7.6c Contours of 10-m interval

7.4.2 Tetrahedral-based volume

Volumes in a region which can be represented by a series of tetrahedra can be easily computed. The generated TEN structure allows such computation. Detailed formulae for the volume computation is presented in Appendix F. This application is useful for calculating volume of 3D objects which can be represented by TEN.

7.5 Height Interpolation for 3D Objects

An interpolation program was developed to facilitate the determination of heights in 3D objects such as buildings so that building's footprints (an area of a building sitting on the terrain surface) could be determined. The following gives the procedure for this interpolation (more detailed explanation is presented in Appendix J (Vol. II).

The purpose of the interpolation is to get the height of the building roof points on the ground surface, i.e. its footprint on the DTM. The interpolation is done by using a point-in-triangle test and a line-triangle plane intersection as illustrated in Figure 7.7.

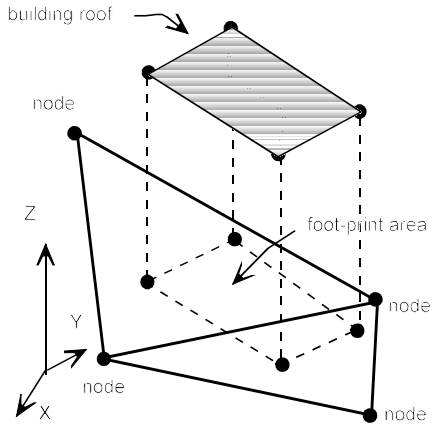


Figure 7.7 Point-in-triangle test and height interpolation for building roof points

The procedure for the interpolation follows:

```
void Interpolation :: MakeInterpolation()
    {
```

```
                // loop from 1st. triangle to last triangle
                    for (int t = 1; t < maxtin; t ++)
                    {
                     Get3Nodes(t, xNd1, yNd1, xNd2, yNd2, xNd3, yNd3);

                     if ( (xroof >= Min3(xNd1, xNd2, xNd3)) &&
                          (xroof <= Max3(xNd1, xNd2, xNd3)) &&
                          (yroof >= Min3(yNd1, yNd2, yNd3)) &&
                          (yroof <= Max3(yNd1, yNd2, yNd3)) )
                        {
                         // point-in-triangle test
                         PointInTriangleTest(xroof, yroof, intri);

                          if (intri == true)
                           {
                      GetPlane(xNd1, yNd1,
                             xNd2, yNd2,
                             xNd3, yNd3,
                             zNd1, zNd2, zNd3,
                             a, b, c, d);
                              // to calculate the interpolated height on the TIN surface (i.e. DTM)
                      zint = ZInTri(xroof, yroof, xNd1, yNd1, zNd1,
                                    xNd2, yNd2, zNd2,
                                    xNd3, yNd3, zNd3);
                      Element.xi = xroof;
                      Element.yi = yroof;
                      Element.zi = zint;

                               // write the interpolated heights to a file
                      AddZtoFile(Element);
                     }
                    }
                }
}
```

This procedure takes a triangle file (.TIN), nodes file (.XYZ) file, and a building's roof coordinates which are also in the form of XYZ format. The building's corner points or other features can be easily acquired using the available 3D digital photogrammetric software (the Helava-Leica's Socet Set) in a photogrammetric workstation. The procedure produces an ASCII file of the interpolated heights for the object's footprints. The same technique is also applied to trees when only photgrammetric data are available. The result of the procedure is illustrated in the 3D perspective view as shown in Figure 7.3.

These newly interpolated footprints coordinates of the buildings will contribute to the creation of 3D TINs. Each TIN can represent a building object. 3D GIS procedures such as volume computation will then be executed.

7.6 Input and Output File Formats

In this work, several file formats were constructed and adopted. The constructed formats are ".XYZ", ".TIN", ".NBR", ".CON", ".SEG", ".ARC", ".ATR". All of these are ASCII files. These are the format for the data input and as well as data output from the developed subsystems as discussed in the foregoing chapters. Adopted file formats are mainly formats which related to the use of the commercial packages such as ILWIS, AVS, and PC Arc/Info. These file formats have the suffix with the three-characters such as ".MPD", ".MPI", ".DAT", and ".LIN". 3D version of raster data and information file for ILWIS also been developed (called .VPD, and .VPI). Detailed description of these formats are given in Appendix D.

7.7 Discussion

The test described in this chapter demonstrates the applicability and workability of the algorithms and other computation tasks (including the tasks discussed in Chapter 6). The display interface both MDI and SDI types work within the specified data input format. Further works are needed to look into:

- improving the types of documents that can be work with.
- having more interface functions such as advance dialogue menu.
- interactive perspective viewing.
- on-the-fly image or document editing.

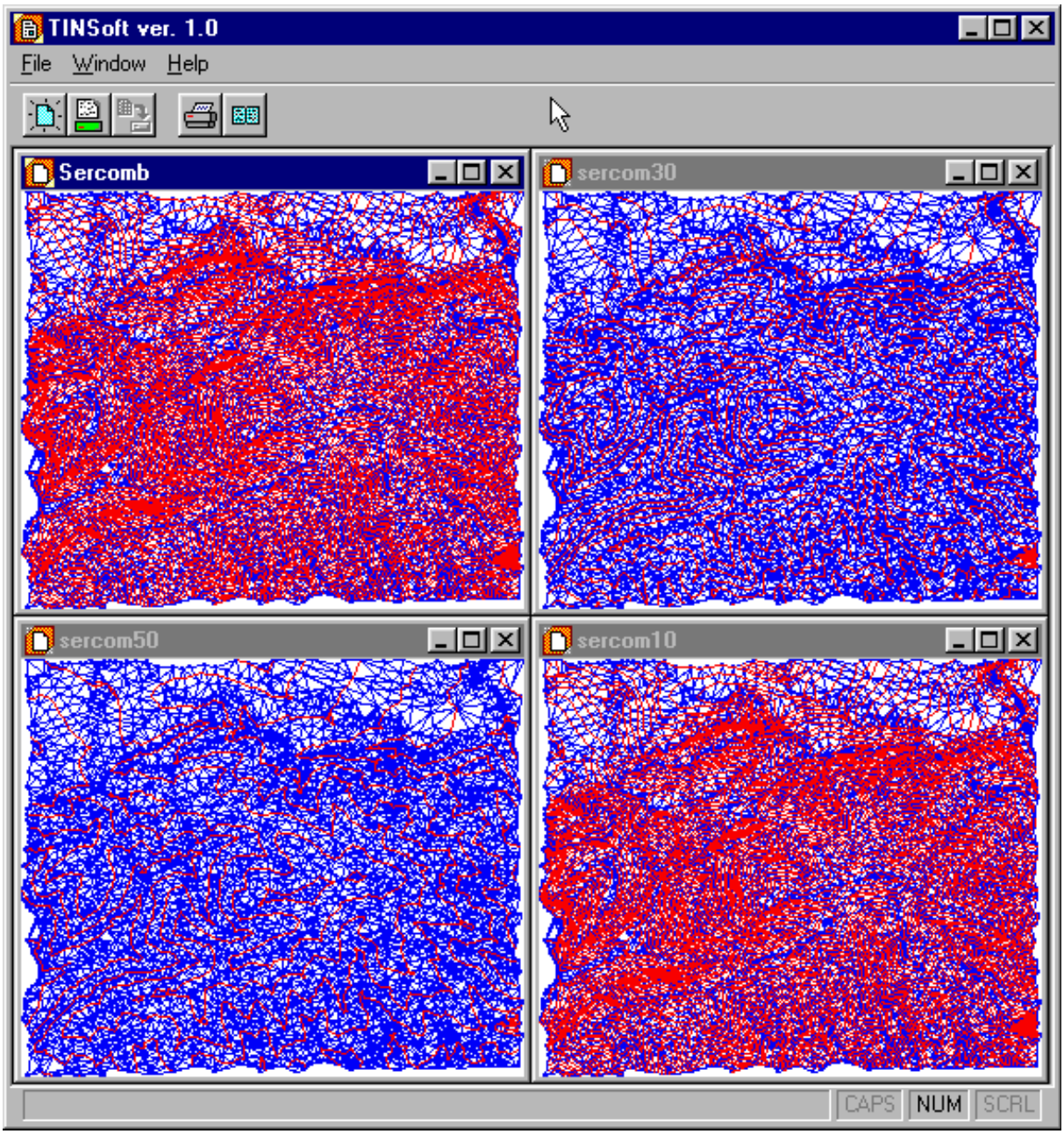


Figure 7.8 Some examples of TINs with the generated derived contours at different intervals.

Chapter 8

Conclusions and Future Research

This chapter provides a general review of the work carried in this research, draws conclusions, and indicates aspects for further research and development of the work investigated in this thesis.

8.1 Summary

First and foremost this work has shown that a GIS can be built using the 2D and 3D TIN (TEN) structures which will correctly represent 2D and 3D objects which fall on and within the earth's surface. Furthermore these representations can be manipulated to provide new information - a requirement of any information system. In more detail the purpose of the research output in this thesis is to improve the handling of 2D and particularly 3D in GIS based on the 2D Triangular Irregular Network (TIN) and 3D TIN (TEN) data structures. The data structuring and the use of an object-oriented technique are the major concern of this work, that is to say an attempt to solve the related research problems as discussed.

Several important aspect of 2D and 3D spatial data structures, data modelling, data construction, database development, and user interface development have been studied and discussed in the last six chapters which include:

- the main problem which lead to the absence of real 3D spatial system in GIS;
- the suitable data structures for 2D and 3D spatial data;
- the relevant aspect of spatial data modelling which contributes to the development of geo information system;
- the modelling of the selected data structures using an object-oriented technique;

- the development of algorithms for the construction of the data structures, and also the relevant applications;
- the implementation and test of the algorithms; and
- graphical user interface.

In chapter 1, a general overview of the problem of 3D data structuring, spatial data modelling and building an information system for 3D spatial data was described. The focus was on using object-oriented techniques for the construction of TIN data structures.

Chapter 2 presented an overview of the 3D GIS development which includes a review of current GIS systems. Four systems from leading software vendors in the GIS market were described. Initially, the chapter also summarised some important functions of a GIS. Major issues in 3D GIS development and it relation to the work carried out in this work are highlighted.

Chapter 3 presented problems related to 2D and 3D spatial data representations. Pertinent data structures were reviewed with an emphasis on selecting the most suitable data structures for the development of a spatial information system. Two types of structures were investigated, the surface-based and the volume-based. Two structures were selected for further work, they were the 2D TIN and 3D TIN (TEN) data structures.

Chapter 4 discussed aspects of fundamental spatial data modelling and how this contributes to spatial information system development. Aspects such as data modelling and spatial database development were also described with emphasis on TIN data where relational database was briefly discussed. GIS and OO TIN-based GIS was also described.

In chapter 5, TIN-based spatial data modelling using object-oriented techniques was

described. The usefulness of the object-oriented approach is discussed. It was not the intention of this chapter to discuss other aspects of object-oriented techniques, for example analysis and design aspects, but rather use them for testing the workability of the developed algorithms. The chapter also provided a possible technique for developing an object-oriented database for TIN data structures using the POET commercial OO DBMS.

Chapter 6 described all the algorithms developed in this research work. There are two main categories of algorithms: for the construction of 2D TIN; and algorithms for the 3D TIN (or TEN). Detailed descriptions of all the functions and methods for the algorithms are provided. Algorithm for TIN-based application is also described, namely contouring. Other applications, that is 3D TIN-based volume computation are described in Chapter 7 with detail formulae in Appendix F. The TIN algorithms were also developed in such a way that linear features which normally exist in the real world can be incorporated. These constrained features extend the type of spatial data in the data modelling, for example a polygon can be generated from the chain of arc (or edge) primitives. Although most of the tests to the algorithms are carried out in this chapter, some other tests are also implemented in Chapter 7.

Chapter 7 described the implementation and test of the algorithms and other spatial computation tasks. The development of user interfaces in a Windows platform is a major concern; the interfaces developed are for the handling multiple views as well as single views. It is not the intention of this work to implement fully a user interface for a GIS, but the development is rather for testing the output of the algorithms as discussed in the previous chapter and to 'set the scene' for subsequent development.

8.2 Discussion

Current research and industrial efforts for solving 3D spatial data information system appears to focus mainly on the non object-oriented approach, that is to say procedural

(structured) techniques and the relational database. This thesis tackles an area of research which utilizes object-oriented techniques for the development of TIN-based spatial data subsystems. The proposed subsystems work as described in chapter 6 and 7. The subsystems represent several of the major components that any GIS would have, that is from data input to a display or visualization subsystem via an analysis subsystem. Although each of the proposed subsystems work and generate good results but they do not operate together as one fully operational system. Aspects of computing such as system integration need to be further considered for this full integration.

All the developed algorithms work and provide a good framework for spatial system development. The performance of the algorithms in terms of computing yard sticks such as speed and data volume is not part of this work.

The following gives the list of the software developed in this work. The software can be categorised into several groups, and their brief operations are summarised in Table 1. Detailed coding of the software is presented in Appendix J (Volume II).

- rasterization module,
- 2D TIN,
- 3D TIN,
- 2D Viewing,
- 3D Viewing,
- POET Database Compatible,
- TIN-based applications, and
- User-interface program.

Table 1 Software documentation

Software	Input	Output
• 2D Rasterization	Point file (.XYZ)	Raster files (.MPI, .MPD)
• Constrained Rasterization	Point file (.XYZ) and arc file (.ARC)	Raster files (.MPI, .MPD)
• 2D Distance Transformation	Raster files of points and lines (.MPI, .MPD).	Raster file of DT image (.MPI, .MPD)
• 2D Voronoi Tessellations	Raster files of DT images (.MPI, .MPD).	Raster files of Voronoi images (.MPI, .MPD).
• 2D TIN Data Structuring	Raster Voronoi images (.MPI, .MPD)	Triangle file (.TIN)
• 3D Rasterization	Point file (.XYZ)	Raster files (.VPI, .VPD). Includes the AVS (.DAT) format.
• 3D Distance Transformation	Raster file (.VPI, .VPD)	Raster 3D DT images (.VPI, .VPD). Includes (.DAT) of AVS format.
• 3D Voronoi Tessellations	Raster 3D DT images (.VPI, .VPD).	Raster 3D Voronoi tessellated images (.VPI, .VPD). Includes (.DAT) of AVS format.
• 3D TIN Data Structuring	Raster 3D Voronoi tessellated images (.VPI, .VPD).	ASCII 3D TIN file (.TEN)
• 2D Viewing	ASCII files of (.XYZ, ARC, .CON)	Windows based softcopy (orthogonal view).
• 3D Viewing	ASCII files of (.XYZ, .TIN, .TEN)	Windows based softcopy (perspective view).
• TIN Data Restructuring	ASCII file of triangles (.TIN)	ASCII file of sides (.SID), and triangles sides (.TRS)

• TIN Topology	ASCII file of triangles (.TIN)	ASCII file of TIN neighbours (.NBR)
• 3D Objects Heights-to-Terrain Interpolation	3D objects height above terrain coords. file (X, Y, Z).	3D objects footprint coords. (on the terrain) file (X, Y, Z).
• Line Segment to Arc/Info Conversion	ASCII file of segment (.SEG)	ASCII file of Arc/Info format (.LIN)
• POET Database Compatible	Database schema files (TNode, TEdge, TPoly, TSolid classes)	Console-based softcopy
• Contouring	ASCII file of (.XYZ, .TIN	ASCII file of contours segments (.SEG)
• User Interface (MDI)	ASCII file of coordinates, triangles, contours.	Windows-based softcopy (Multiple views).

8.3 Conclusions

Based on the research reported in the last seven chapters, the following conclusions can be drawn:

- Data structures of type 2D TIN and 3D TIN (TEN) are able to support several important spatial data primitives for the development of a spatial system. The primitives of node, edge, polygon and solid (body) form important aspects of spatial data modelling. The 2D TIN data structure proved to be useful for representing point, linear, and area features. These spatial data representations have nodes, and edges as been described in Chapter 3.

- The usefulness of the 2D TIN data structure is improved by the development and implementation of constrained triangulations. Various types of terrain features can be incorporated into the spatial data base by having this constrained feature

in the triangulation subsystem. The two subcomputations for the triangulation proved to work, namely distance transformation (DT), and Voronoi tessellations.

- Although few applications have been completed, the generated 3D TIN (or TEN) data structure can be used to handle 3D data for example nodes surveyed during boreholing. The relationship between nodes along several boreholes can be represented by the TEN structure to provide a representation of solids (e.g. ore bodies). Volume using the TEN structure for underground regions can be determined by executing the POET compatible program developed in this thesis (see Vol. 2, "POET Database Compatible Program").

- Object-orientation plays a useful role in data modelling. The facilities provided in an OO programming language, such as inheritance, classification and encapsulation are useful tools for system design and implementation. The object-oriented approach for 2D and 3D TIN proved beneficial especially in the 're-use' capability thus reducing programming effort. Existing methods associated with a class of a domain, for example the Voronoi tessellation program can utilise some of the methods developed for other classes such as the DT class.

- The generated data structures can be use to define several important GIS data types: point, line, polygon and solid. These GIS data types form the four major classes of the OO spatial data modelling and OO database development.

- The POET OO database management system can be used for the development of the TIN-based database. The generated database can be browsed and queried using the built-in object query language (OQL).

- The tests on the algorithms clearly indicate that all of the proposed subsystems functioning and providing promising output. The user interfaces also functioning and their usefulness is illustrated.

- Application programs such as contour derivation and volume computation function. More related applications (particularly 3-dimensional) can be investigated in the future to diversify the usefulness of the subsystems developed in this research work.

8.4 Future Research

As described in the last sections, this research has covered a number of aspects of 2D and 3D spatial data structuring, data modelling, database populating, application development and user interface for a GIS. An important component of this research is the software that can be used for the development of an operational GIS system. Key aspects and problems were identified and have been implemented and tested, thus it can be concluded that the objectives set out in Chapter 1 have been achieved. However, there are still other related issues that need to be investigated further and considered for future development. The following is a summary for future work:

- Implementing the constrained 3D TIN; incorporating 3D constrained features will extend data handling and provide more spatial information.

- Further developing and formalising 3D spatial data. This is an important task for describing the relationships and links between data and is the next logical step in the modelling process.

- Integrating 2D and 3D TIN data structures into one common subsystem. In this work, they were constructed separately.

- Developing and implementing a graphic file format for the DT and Voronoi tessellation images, avoiding the use of external packages.

- Redesigning the OO spatial data model using object-oriented data analysis and

design tools such as UML (United Modelling Language) to accommodate more complex situations.

- Developing and implementing spatial operators for spatial database manipulations.

- Constructing an advanced graphics user interface as the front-end program. Simple display interfaces have been developed. Although they were able to perform the tasks further work needs to be considered.

And finally,

- Investigating an optimal integration between the developed subsystems with an object-oriented database management system (OO DMBS) as a database engine for facilitating 2D and 3D TIN-based GIS.

It is hoped that the author will be able to address some of these topics in his future research on his return to his own academic department in Malaysia.

References and Bibliography

1. ABDUL-RAHMAN, A. (1992). Triangular irregular networks in digital terrain relief modelling. *M.Sc. Thesis*, ITC, Enschede, The Netherlands, 80 p

2. ABDUL-RAHMAN, A. (1999). Spatial tessellations using an object-oriented approach. *Proceedings of GIS Research UK (GISRUK' 99),* April, Southampton, England.

3. ABDUL-RAHMAN, A., and DRUMMOND, J. E. (1998). Raster-based algorithms for 2D and 3D TINs generations. *Proceedings of International Conference of Spatial Information Science and Technology (SIST' 98),* December, Wuhan, China, 9 p

4. ABDUL-RAHMAN, A., DRUMMOND, J. E., and SHEARER, J. W. (1998). Representation of 3D Spatial Objects, *American Congress on Surveying and Mapping (ACSM) Annual Conference and Exhibition*, 2-4 March, Baltimore, USA.

5. ABDUL-RAHMAN, A., and DRUMMOND, J. E. (1999). The development of 2D and 3D triangular irregular networks computer programs. *Proceedings of American Congress on Surveying and Mapping (ACSM) Conference and Exhibition*, Portland, Oregon. 13-17 March, pp. 38-49.

6. ABDUL-RAHMAN, A., and DRUMMOND, J. E. (2000). The implementation of object-oriented TIN-based subsystems for GIS. *Proceedings of International Society of Photogrammetry and Remote Sensing (ISPRS) Congress,* Vol. IV, CD ROM 1, July, Amsterdam, The Netherlands.

7. ANTENUCCI, J. C., BROWN, K., CROSWELL, P. L., KEVANY, M. L., ARCHER, H. (1991). Geographic information systems, a guide to the technology. New York, Van Nostrand Reinhold, 301 p

8. AVS (1997). Advance Visualization System. Version 3.1. (http://www.avs.com)

9. AYBET, J. (1992). Object-oriented GIS: what dose it mean to GIS users. *Proceedings of European Geographic Information Systems (EGIS)*, Session 50, pp. 1279-1287.

10. BERG, M. de (1997). Visualization of TINs, In: *Algorithmic Foundations of Geographic Information Systems, Lecture Notes in Computer Science*, No. 1340, (eds.) van Kreveld, M., Nievergelt, J., Roos T., and Widmayer P., Springer, pp. 79-97.

11. BHALLA, N. (1991). Object-oriented data models: a perspective and comparative review. *Journal of Information Science*, Vol. 17, pp. 145-160.

12. BONHAM-CARTER, G. F. (1996). Geographic information systems for geoscientists: modelling with GIS. *Computer Methods in the Geosciences*, Vol. 13, Pergamon Publications, 398 p

13. BOOCH, G. (1990). Object oriented analysis and design with applications. 2nd. Edition, Addison-Wesley, 589 p

14. BORGEFORS, G. (1986). Distance transformations in digital images. *Computer Vision, Graphics, and Image Processing (CVGIP)*, Vol. 34, Academic Press, pp. 344-371.

15. BORGEFORS, G. (1996). On digital distance transformations in three dimensions. *Computer Vision and Image Understanding (formally known CVGIP)*, Vol. 64, No. 3, pp. 368-376.

16. BORLAND. (1996). Borland™ Object Windows version 5.0 programmers guide. *Borland International Inc*., 538 p

17. BRIC, V. (1993). 3D vector data structures and modelling of simple objects in GIS. *M. Sc. Thesis*, ITC, Enschede, The Netherlands, 107 p

18. BRIC, V., PILOUK, M., and TEMPFLI, K. (1994). Towards 3D-GIS: Experimenting with a Vector Data Structure. *Proceeding of the Symposium on Mapping and Geographic Information Systems*, Georgia, USA, ISPRS Archives Vol. 30, Part 4, pp. 634-640.

19. BRUNET, P. (1992). 3D structures for encoding of geometry and internal properties. In: *Three-Dimensional Modelling with Geoscientific Information Systems* (eds.) A. K.

Turner, NATO ASI Series, Vol. 354, pp. 159-188.

20. BURROUGH, P. A. and FRANK, A. (1995). Geographic objects with indeterminate boundaries. Taylor & Francis, GISDATA 2 Series, 345 p

21. BURROUGH, P. A., and McDONNELL, R. A. (1998). Principles of geographical information systems. Oxford University Press, 333 p

22. CAMBRAY, de B.(1993). Three-dimensional (3D) modelling in a geographical database. *Proceedings of AutoCarto 11*, Minneapolis, USA, pp. 338-347.

23. CAMBRAY, de B., and YEH, T. S. (1994). A multidimensional (2D, 2.5D, and 3D) geographical data model. *International Conference on Management of Data (COMAD'94)*, Bangalore, India, Tata Mc Graw-Hill, pp. 317-336. (http: //www.prism.uvsq.fr/rapports/1994/)

24. CARLSON, E. (1987). Three dimensiponal conceptual modelling of subsurface structures. *Technical Papers of ASPRS/ACSM Annual Convention*, Vol. 4, Baltimore, pp. 188-200.

25. CHEN, H. W.(1991). The implementation of simultaneous octree generation for handling 3D geographic region information in a GIS. *M. Sc. Thesis*, ITC, Enschede, The Netherlands, 43 p

26. CHOU, Y. H. (1996). Exploring spatial analysis in geographic information systems. Onword Press, Santa Fe. USA. 474 p

27. DELAUNAY, B. (1934). Sur la sphére vide. *Bulletin of the Academy of Science if the USSR, Classe. Sci. Mat*., pp. 793-800.

28. DELOBEL, C., LECLUSE, C., and RICHARD, P. (1995). Databases: from relational to object-oriented systems. International Thomson Computer Press, London, 382 p

29. DONG, FENG (1996). Three-dimensional models and applications in subsurface modelling. *M. Sc. Thesis*, University of Calgary, Canada, 93 p

30. EGENHOFER, M. and FRANK, A. U. (1989). Object-oriented modelling in GIS: inheritance and propagation. *Proceedings of AutoCarto 9*, Baltimore, pp. 588-598.

31. ERDAS (2000). IMAGINE VirtualGIS. ERDAS Inc. publication, USA. (http://www.erdas.com)

32. ESRI (1997). Using ArcView 3D Analyst. Environmental System Research Institute (ESRI) Publication, Redlands, California, USA, 118 p

33. ESRI (2000). ArcView GIS. Environmental System Research Institute (ESRI) Publication, Redlands, California, USA. (http://www.esri.com)

34. FARRELL, J. A. (1994). From pixels to animation: an introduction to graphics programming. AP Professional, 676 p

35. FOLEY, J. D, van DAM, A., FEINER, S. K., and HUGHES, J. F. (1996). Computer graphics: Principles and Practices, 2nd. Edition in C., Addison-Wesley Publishing, 1174 p

36. FÖRSTNER, W. (1995). GIS - the third dimension. Workshop "Current Status and Challenges of Geoinformation Systems", IUSM Working Group on LIS/GIS, University of Hannover, Germany, pp. 65-72.

37. FORTUNE, S. (1992). Voronoi diagrams and Delaunay triangulations. In: D. Z. Du and F. Hwang (eds.), *Computing In Euclidean Geometry, Lecture Notes Series on Computing* - Vol. 1, World Scientific Publishing, pp. 193-233.

38. FRANK, A. U., and KUHN, W. (1986). Cell graphs: a provable correct method for the storage of geometry. *Proceed. of 2nd. International Symposium on Spatial Data Handling*, Seattle, Washington, USA., pp. 411-436.

39. FRITSCH, D. (1996a). The integration of spatial object models and DTM, 2.5D and 3D models. *International Society of Photogrammetry and Remote Sensing (ISPRS) Congress Tutorial*, Vienna Technical University, Austria.

40. FRITSCH, D. (1996b). Three-dimensional geographic information systems - status and prospects. *International Archives of Photogrammetry and Remote Sensing (IAPRS)*, Vol. 31, Part 4, K. Kraus and P. Waldhausl (eds.), XXXI International Congress of Photogrammetry and Remote Sensing, Vienna, Austria, pp. 215-221.

41. FRITSCH, D. and SCHMIDT, D. (1995). The object-oriented DTM in GIS. *Proceedings of Photogrammetric Week*, Stuttgart, pp. 29-34.

42. GEOMATICS. (2000). PMAP GIS Product. PCI Geomatics Group. (http://www.pcigeomatics.com)

43. GOLD, C. M., REMMELE, P. R., and ROOS, T. (1997). Voronoi methods in GIS. In: M. van Kreveld, J. Neivergelt, T. Roos, and P. Widmayer (eds.), *Algorithmic Foundations of Geographic Information Systems*, Lecture Notes in Computer Science, No. 1340, Springer-Verlag, Berlin, pp. 21-35.

44. GORTE, B., and KOOLHOVEN, W. (1990). Interpolation between isolines based on the Borgefors distance transform. *ITC Journal*, 1990-3, pp. 245-247.

45. GUO, WEI. (1996). Three-dimensional representation of spatial object and topological relationships, *International Archives of Photogrammetry and Remote Sensing (IAPRS)*, Vol. XXXI, Part B3, Commission 3, K. Kraus and P. Waldhausl (eds.), XXXI International Congress of Photogrammetry and Remote Sensing, Vienna, Austria, pp. 273-278.

46. HEALEY, R. G. (1991). Database management systems. In: Maguire D. J., Goodchild M. F., Rhind D, W. (eds.) *Geographical Information Systems: Principles and Applications*, Longman/New York, John Wiley & Sons Inc, Vol. 1, pp. 251-267.

47. HEITZINGER, D, and PFEIFER, N. (1996). A new approach to modelling 3D surfaces. In: Advanced DTM Technology, *ISPRS Congress Tutorial*, Technical University of Vienna, Austria.

48. HOULDING, S. W. (1994). 3D geoscience modelling: Computer techniques for geological characterization. Springer-Verlag, Berlin, 309 p

49. ILWIS, (1996). Integrated Land and Watershed Information System. Version 2.1, ITC. The Netherlands.
(http://www.itc.nl/ilwis)

50. INTERGRAPH (2000). GeoMedia Product. Intergraph Inc.
(http: //www.intergraph.com)

51. JONES, C. B.(1989). Data structures for three-dimensional spatial information systems in geology. *International Journal of Geographic Information System (IJGIS)*, Vol. 1, no. 3, pp. 15-31.

52. KHOSHAFIAN, S. and ABNOUS, R. (1995). Object orientation: concepts, analysis and design, languages, databases, graphical user interfaces, and standards. 2nd. Edition, John Wiley & Sons Inc., 504 p

53. KRAAK, M. J. (1992). Working with triangulation-based data in 3D space. *ITC Journal*, 1992-1, pp. 20-24.

54. KRAUS, K. (1995). From digital elevation model to topographic information system. *45th.Photogrammetric Week*, D. Fritsch and D. Hubbie (eds.), Stuttgart, pp. 277-285.

55. KUFONIYI, O. (1995). Spatial coincidence modelling, automated database updating and data consistency in vector GIS. ITC Publication No. 28, Enschede, The Netherlands, 201 p

56. LANGRAN, G. (1992). Time in geographic information systems. Taylor & Francis Publication, 189 p

57. LAURINI, R., and THOMPSON, D. (1991). Fundamentals of spatial information systems. Academic Press, 680 p

58. LI, QINGQUAN. and LI, DEREN (1996). Hybrid data structures based on octree and tetrahedron in 3D GIS. *International Archives of Photogrammetry and Remote Sensing (IAPRS)*, Vol. XXXI, Part B, Comm. 4, ISPRS Congress, Vienna, pp. 503-507.

59. LI, RONGXING (1994). Data structures and application issues in 3D geographic

information systems. *Geomatica*, Vol. 48, No. 3, pp. 209-224.

60. LI, RONGXING, CHEN, Y., DONG, F., QIAN, L., and HUGHES, J. D. (1996). 3D data structures and applications in geological surface modelling. *International Archive of Photogrammetrry and Remote Sensing (IAPRS)*, Vol. XXXI, Part B4, K. Kraus and P. Waldhausl (eds.), pp. 508-513.

61. LIMP, W. F. (1999). ERDAS Imagine VirtualGIS 8.4. Adams Business Media Inc. (http://www.gw.geoplace.com/bg/1999/productreview/499qt.asp)

62. LONGLEY, P.A., GOODCHILD, M. F., MAGUIRE, D. J., and RHIND, D.W. (1999). Geographical information systems: principles and technical issues. (Vol. 1.). 2nd. Edition, John Wiley & Sons, 580 p

63. MAGUIRE, D. J. (1999). GIS customisation. In: Geographical Information Systems: principles and technical issues, Vol. 1, (eds.) Paul A. Longley, M. F. Goodchild, D. J. Maguire, D. W. Rhind, pp. 359-369.

64. MÄNTYLÄ, M.(1988). Introduction to solid modelling. Computer Science Press, Maryland, USA, 401 p

65. MARK, D. M. and CEBRIAN, J. A. (1986). Octrees: a useful data-structure for processing of topographic and sub-surface data. *Technical Papers of ACSM-ASPRS Annual Convention*, Vol. 1 (Cartography and Education).

66. McCULLAGH, M. J., and ROSS C. G. (1980). Delaunay triangulation of a random data set for isorithmic mapping. *The Cartographic Journal*, Vol. 17, No. 2, pp. 93-99.

67. MEAGHER, D.(1982). Geometric modelling using octree encoding. *Computer Graphics and Image Processing*, Vol. 19, pp. 129-147.

68. MIDTBØ, T.(1996). Spatial modelling by delaunay networks of two and three dimensions. *Dr. Ing. Thesis*, Norwegian Institute of Technology, University of Tronheim, Norway.
(http://www.iko.unit.no/tmp/)

69. MIRANTE, A., and WEINGARTEN, N. (1982). The radial sweep algorithm for constructing triangulated irregular networks. *IEEE Transaction on Computer Graphics and Applications*, Vol. 2, No. 3, pp. 11-21.

70. MOLENAAR, M. (1989). Single valued vector maps - a concept in GIS. *Geo-Information-Systeme*, Vol. 2, No. 1, pp. 18-26.

71. MOLENAAR, M. (1991). Terrain objects, data structures and query spaces. In: *Geoinformatik* (eds. P. A. Schilcher), Siemens-Nixdorf Informationssyteme A.G., Munchen, Germany, pp. 53-70.

72. MOLENAAR, M. (1992). A topology for 3D vector maps. *ITC Journal*, 1992-1, pp. 25-33.

73. MOLENAAR, M. (1996a). An introduction to theory of spatial object modelling for GIS. Taylor & Francis Publication, London, 246 p

74. MOLENAAR, M. (1996b). Spatial data modelling for geographic information systems. *International Society of Photogrammetry and Remote Sensing (ISPRS) Congress Tutorial*, Vienna Technical University, Austria.

75. PETRIE, G. and KENNIE, T. J. M. (1990). Terrain modelling in surveying and civil engineering. Whittles Publishing, Glasgow, 351 p

76. PEUCKER, T. K., R. J. FOWLER, R. J., LITTLE, J. J., and MARK, D. M. (1978). The triangulated irregular network. *Proceedings of Digital Terrain Modelling (DTM) Symposium, American Society for Photogrammetry (ASP)*, St. Louis, Missouri, pp. 516-540.

77. PEUCKER, T. and CHRISMAN, N. (1975). Cartographic data structures. *The American Cartographer*, Vol. 2, No. 2, pp. 55-69.

78. PEUCKER, T. (1978). Data structures for digital terrain models: discussion and comparison. *1st. International Advanced Study Symposium on Topological Data Structures for Geographical Information Systems*, Harvard Paper on GIS, (edt.) by G. Dutton, Vol. 5.

79. PILOUK, M. (1996). Integrated modelling for 3D GIS. *PhD Thesis*, Wageningen Agricultural University and ITC, Publication No. 40, ITC, Enschede, The Netherlands, 200 p

80. POET. (1996). POET™ Release 4. Manual, POET Inc., 560 p. (http://www.poet.com)

81. RAPER, J. (1990). The 3-dimensional geoscientific mapping and modelling system: a conceptual design. In: *Three Dimensional Applications in Geographic Information Systems*, J. Raper (ed.), Taylor & Francis, pp. 11-19.

82. RAPER, J. (1992). Key 3D modelling concepts for geoscientific analysis. In: *Three Dimensional Modelling with Geoscientific Information Systems,* (ed. A. K. Turner), Series C: Mathematical and Physical Sciences, Vol. 354, Kluwer Academic Publishers, The Netherlands, pp. 251-232.

83. RAPER, J. and MAGUIRE, D. J (1992). Design models and funtionality in GIS. *Computers & Geosciences*, Pergamon Press, Vol. 18, No. 4, pp. 387-394.

84. RAPER, J. and KELK, B. (1991). Three-dimensional GIS. In: *Geographical Information Systems: Principles and Applications.* D. J., Maguire, M. Goodchild and D. Rhind (edts.), Longman Geoinformation, pp. 219-317.

85. RIMSCHA, M. V. (1997). 3d or not 3d?. *GIS Europe*, Pearson Professional Ltd Publication, Issue 10 (October), pp.20-21.

86. ROSENFELD, A., and PFALTZ, J. (1966). Sequential operations in digital picture processing. *Association of Computing Machine (ACM) Journal*, Vol. 13, pp.471-494.

87. ROSS, T. J., WAGNER, L. R., and LUGER, G. F. (1992). Object-oriented programming for scientific codes: thoughts and concepts. *Journal of Computing in Civil Engineering*, Vol. 6, No. 4, pp. 480-514.

88. SAMET, H. (1984). The quadtree and related hierarchical data structures. *ACM Computing Surveys*, Vol. 16, No. 2, pp. 187-260.

89. SAMET, H. (1990). Applications of spatial data structures. Addison-Wesley, 507 p

90. SCHREFL, M., and BICHLER, P., (1995). Modelling reality in spatial information system. In: *Geographic Information Systems - Materials for a Post-Graduate Course*, Dept. of Geoinformation, Technical University of Vienna (ed. A. Frank), pp. 35-89.

91. STROUSTRUP, B. (1997). The C++ programming language. 3rd. Edition. Addison-Wesley, 910 p

92. TANG, L. (1992). Raster algorithms for surface modelling. *International Archives of Photogrammetry and Remote Sensing (IAPRS)*, Vol. XXIX, Comm. 3. Washington D.C., pp. 566-573.

93. TURNER, A. K. (1992a). Report of closing discussions. Workshop on Three-Dimensional Modelling with Geoscientific Information Systems. NATO ASI Series C (Vol. 354), pp. 401-410.

94. TURNER, A. K. (1992b). Three-dimensional modelling with geoscientific information systems. NATO ASI Series C (Vol. 354), 443 p

95. van KREVELD, M. (1997). Digital elevation models and TIN algorithms, In: *Algorithmic Foundations of Geographic Information Systems,* (Eds.) M. van Kreveld, J. Nievergelt, T. Roos, and P. Widmayer, Lecture Notes in Computer Science, No. 1340, Springer, Berlin, pp. 37-78.

96. WACHOWICZ, M. (1999). Object-oriented design for temporal GIS. Taylor & Francis, 118 p

97. WATSON, D. F. (1981). Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes. *The Computer Journal,* Vol. 24, No. 2, pp. 167-172.

98. WEBSTER, C. (1990). The object-oriented paradigm in GIS. *International Archive of Photogrammetry and Remote Sensing (IAPRS)*, Vol. 28, Part 3/2, Comm. III, Wuhan, China, pp. 947-984.

99. WORBOYS, M. F. (1995). GIS: a computing perspective. Taylor and Francis Publication, 376 p

100. WORBOYS, M. F, (1999). Relational databases and beyond. In: *Geographical Information Systems, Vol. 1, 2nd. Edition*, (eds. P.A. Longley, M. F. Goodchild, D. J. Maguire, and D. W. Rhind), John Wiley, pp. 373-384.

101. ZEITOUNI, K., and CAMBRAY, B. de. (1995). Topological modelling for 3D GIS. *4th. International Conference on Computers in Urban Planning and Urban Management*, Melbourne, Australia.

102. ZLATANOVA, S. (2000). 3D GIS for urban development, *PhD thesis*, ITC (The Netherlands) and Technical University of Graz (Austria).

Appendix A

2D and 3D Rasterization

A1.0 Introduction

Spatial data can be categorized as vector and raster. Each data type has its own advantages and disadvantages as mentioned in various cartography and GIS textbooks. To ease some spatial data processing and manipulation tasks, a conversion of data from one form to the other is needed. The conversion from vector to raster is called rasterization. Rasterization is usually a process of converting a world coordinate image, vector to grid form. The devices could be computer screen, plotter, scanner, and others. To facilitate the development of 2D and 3D TIN, two rasterization programs have been developed. The output of the programs is input to 2D and 3D distance transformation, Voronoi tessellation and triangulation.

A2.0 The rasterization formulae

Two steps are involved in rasterization. The steps are scaling and translation. It is like a conformal transformation. In order to use all the nodes in the data sets, every point needs to be rasterised and each point represented by one raster cell. To ensure no nodes are the same or in adjacent pixels the minimum distance between two nodes has to be known before rasterization. Then, a scale factor for the rasterization should be determined by taking a ratio of 1 over the smallest distance between the two nodes, see Figure A1.1. The distance may be determined either by computing from the data sets or by visualizing and measuring on a screen with an appropriate module.

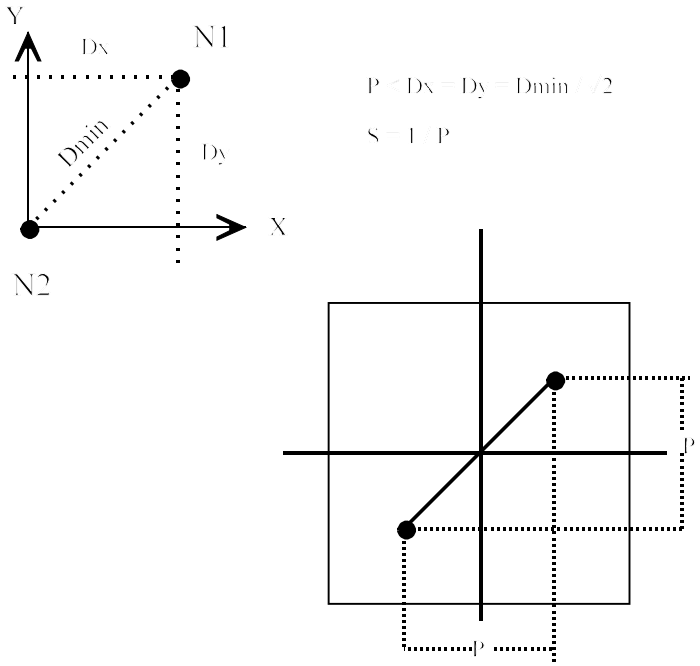


Figure A1.1 Scale factor determination for rasterization

This measure will eliminate the possibility of having duplicated nodes in one pixel cell or two in adjacent cells. Figure A1.1 shows how we could determine the scale factor for the rasterization.

Having determined the scale factor, then the array dimensions for pixels, i.e. the extent of rows and columns can be determined as follows:

$Dim_x = (X_{max} - X_{min}) * S$
$Dim_y = (Y_{max} - Y_{min}) * S$
$Dim_z = (Z_{max} - Z_{min}) * S$

where,

Dim_x, Dim_y, and Dim_z are array dimensions for X, Y, and Z respectively,
X_{max}, Y_{max}, Z_{max} are the maximum extents for the X, Y, Z coordinates from data sets, and
X_{min}, Y_{min}, Z_{min} are the minimum extents for the X, Y, and Z coordinates from the same data sets.

In the case of 2D, the (X, Y) coordinates were used, and the transformation formulae in the form of a matrix may be written as follows:

$$\begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} a_{11} & a_{21} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix} + \begin{bmatrix} T_i \\ T_j \end{bmatrix}$$

where,

i, j = indices of the array,
a_{mn} = elements of transformation matrix,
X, Y = world coordinate tuple, and
T_i, T_j = translation vector.

During the rasterization, all the (X, Y) world coordinates need to be translated to the origin (0, 0) and this is done by subtracting the X_{min}, and Y_{min} values of the data sets from the coordinates. Since the origin of the device (i.e. a computer screen) is located at the top left corner, then all the Y coordinates need to be multiplied with negative 1 so that the positive y-direction from top to bottom can be facilitated. Thus, give the following equation for the rasterization:

$$\begin{bmatrix} i \\ j \end{bmatrix} = \text{truncate}\left(\begin{bmatrix} S_x & 0 \\ 0 & -S_y \end{bmatrix} \begin{bmatrix} X - X_{min} \\ Y - Y_{min} \end{bmatrix} \right) + \begin{bmatrix} 1 \\ j_{max} + 1 \end{bmatrix}$$

where,

S_x, S_y = scale factor for X and Y respectively,
X_{min}, Y_{min} = minimum coordinate from data sets,
j_{max} = maximum row of the raster image.

After having calculating the index for every X, Y of the nodes, then the next task is to assign the identification code (id) for every point that is being transformed . The id of the points thus assigned to the nodes are as follows:

A(i, j) = id of P(X, Y)

For the 3D (i.e. nodes with X Y, Z coordinates), a similar technique is applied. The corresponding matrix form for the transformation follows:

$$\begin{bmatrix} i \\ j \\ k \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + \begin{bmatrix} T_i \\ T_j \\ T_k \end{bmatrix}$$

where,

i, j, k = indices of the array,
a_{mn} = elements for the transformation matrix,
X, Y, Z = the world coordinates tuple, and
T_i, T_j, T_k = translation vector.

As in the case of 2D, the (X, Y, Z) coordinates need to be translated to the origin of the devices (i.e. 0, 0, 0). This could be done by subtracting from the coordinates the X_{min}, Y_{min}, and Z_{min} values of the data sets. Then, the corresponding rasterization formulae for 3D rasterization could be written as follows:

$$\begin{bmatrix} i \\ j \\ k \end{bmatrix} = \text{truncate}\left(\begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & -S_z \end{bmatrix} \begin{bmatrix} X - X_{min} \\ Y - Y_{min} \\ Z - Z_{min} \end{bmatrix} \right) + \begin{bmatrix} 1 \\ 1 \\ k_{max} + 1 \end{bmatrix}$$

where,

S_x, S_y, S_z = scale factor for X, Y, Z respectively,
X_{min}, Y_{min}, Z_{min} = minimum coordinates of data sets, and
k_{max} = maximum dimension of the array for the third dimension's rows.

After having determined the array indexes for every point, the identifier (id) is then assigned to this array element by

e.g. V(i, j, k) = id of point(X, Y, Z).

Class Definitions

The following are the definitions of the classes used for the 2D TIN data construction.

The Distance Transformation (DT) class:

```
//*************************************************************//
//                                                             //
// An Object-Oriented Program: Distance Transformation (DT)    //
//  Input: MPD and MPI file (rasterised points files)          //
//  Output: MPD and MPI file(DT raster files)                  //
//                                                             //
//*************************************************************//

// File: TDistanceTransform.h

#include <iostream.h>
#include <stdlib.h>
#include <stdio.h>
#include <fstream.h>
#include <io.h>
#include <iomanip.h>
#include <malloc.h>
#include <conio.h>
#include <limits.h>
#include <stddef.h>

#define masksize 8

class TDistanceTransform
 {
  public:

 // member data
     typedef struct MpiStruct
       {
        short Nscanlines;
        short Npixels;
        short Pvmin;
        short Pvmax;
```

```
        short Maptype;
        short Patch;
        short Scale;
        short Crdtype;
        float a11;
        float a12;
        float a21;
        float a22;
        float b1;

float b2;
      } MpiType;

    typedef short Image;
    typedef Image* ImagePtr;
    typedef ImagePtr* ImagePPtr;
    typedef short Mask[9];

    MpiType MPI, MpiOut;
    Mask MaskPix;
    int row;
    int col;
    int r;
    ImagePPtr Pixel;
    int maxrow;
    int maxcol;
    FILE* MPIfile;
    FILE* MPDfile;
    FILE* MPIofile;
    FILE* MPDofile;


  // member functions
    void GetMPIInputFile();
    void GetMPDInputFile();
    void ReadImage(ImagePPtr& Pixel);
    void PrintPixel(ImagePPtr Pixel);
    void WriteMPIOutputFile();
    void WriteMPDOutputFile();
    void SetBackground(ImagePPtr Pixel, int Bg, int Fg);
    void GetUpperMask(int r, int c, ImagePPtr Pixel, Mask& MaskPix);
    void GetLowerMask(int r, int c, ImagePPtr Pixel, Mask& MaskPix);
    int MinByIndex(int from, int to);
    int Min5(int a, int b, int c, int d, int e);
    void DistancePassOne(ImagePPtr Pixel);
    void DistancePassTwo(ImagePPtr Pixel);
```

```
    void ForwardDistance();
    void BackwardDistance();
    void PressAnyKey();
    void Title();
    TDistanceTransform();        // constructor
    ~TDistanceTransform();       // destructor

};
```

The Voronoi Tessellation class:

```
//*************************************************************************//
//          A Voronoi Tesselation program for 2D TIN                       //
//          Input  :  MPI and MPD files of DT program                      //
//          Output :  MPI and MPD files                                    //
//                                                                         //
//*************************************************************************//

// File: TVoronoi.h


#include <iostream.h>
#include <stdlib.h>
#include <stdio.h>
#include <fstream.h>
#include <io.h>
#include <iomanip.h>
#include <dos.h>
#include <alloc.h>
#include <malloc.h>
#include <conio.h>
#include <limits.h>
#include <stddef.h>

#include "TDistanceTransform.h"

#define masksize 8                  // Mask size

class TVoronoiTessellation : public TDistanceTransform
 {
  public:
    TVoronoiTessellation();          // constructor
    ~TVoronoiTessellation();         // destructor

   Mask MaskPixV;
```

```
    FILE* MPDfile;
    FILE* MPDodfile;
    FILE* MPDovfile;
    FILE* MPIodfile;
    FILE* MPIovfile;
    ImagePPtr PixelV;

    // Function members
    void GetMPDInputFile();
    void ReadImage(ImagePPtr&);
    void CopyImage(ImagePPtr, ImagePPtr&);
    void GetMPIInputFile();
    void WriteMPIOutputFileDist();
    void WriteMPDOutputFileDist();
    void WriteMPIOutputFileVoronoi();
    void WriteMPDOutputFileVoronoi();
    void SetBackground(ImagePPtr, int, int);
    void GetUpperMaskDist(int, int, ImagePPtr, Mask&);
    void GetLowerMaskDist(int, int, ImagePPtr, Mask&);
    void GetUpperMaskVoronoi(int, int, ImagePPtr, Mask&);
    void GetLowerMaskVoronoi(int, int, ImagePPtr, Mask&);
    int MinByIndex(int, int);
    void ForwardPass(ImagePPtr, ImagePPtr);
    void BackwardPass(ImagePPtr, ImagePPtr);
    void ForwardVoronoi();
    void BackwardVoronoi();
    void PressAnyKey();
    void Title();
};
```

The TINGeneration class:

```
//*************************************************************************//
//            Object-Oriented Voronoi-to-TIN program                       //
//            Input  :  MPI and MPD file of DT program                     //
//            Output :  MPI and MPD files                                  //
//                                                                         //
//*************************************************************************//

#include <iostream.h>
#include <stdlib.h>
```

```
#include <stdio.h>
#include <fstream.h>
#include <io.h>
#include <iomanip.h>
#include <dos.h>
#include <alloc.h>
#include <malloc.h>
#include <conio.h>
#include <limits.h>
#include <stddef.h>

#include "TVoronoi.h"
//#include "TDistanceTransform.h"

// Constant section

#define masksize4 4              // 2 x 2 Mask size

class TTinGeneration : public TVoronoiTessellation
 {
  public:
    TTinGeneration();          // constructor
    ~TTinGeneration();         // destructor

    typedef short DataType;
    typedef struct VertexStruct
            {
             DataType N1;
             DataType N2;
             DataType N3;
            } TVertex;

    typedef struct TsNodeStruct
            {
             short x;
             short y;
            } TsNode;

    // data members
    typedef short Image;
    typedef Image* ImagePtr;
    typedef ImagePtr* ImagePPtr;
    typedef short Mask[4];


    //MpiType MPI;
```

```
    Mask MaskPix;
    ImagePPtr Pixel;
    FILE* MPDfile;
    FILE* MPIfile;

    FILE* TINfile;
    int i, r;
    int NTri;
    TVertex Element;
    TsNode TsPnt;
    bool FoundTri;

    // function members
    void GetMPDfile();
    void ReadImage(ImagePPtr&);
    void GetMPIfile();
    void GetTINfile();
    bool Less(DataType, DataType);
    bool Greater(DataType, DataType);
    void Swap(DataType&, DataType&);
    void NodeOrder(DataType&, DataType&, DataType&);
    void AddTritoFile(TVertex);
    void GetMask(int, int, ImagePPtr, Mask&);
    void GetSubImage(int, int, ImagePPtr, Mask&);
    void ScanlinesUp(Mask);
    void ScanlinesDown(Mask);
    void Scanlines(ImagePPtr);
    void MakeTIN();
    void Title();
    void PressAnyKey();
};
```

The TINView class:

```
//*********************************************************//
// A Windows program for TIN display                       //
//   Input: XYZ coordinates and TIN file                   //
//   Output: TINs in Windows environment                   //
//*********************************************************//

#include <owl/pch.h>
#include <owl/applicat.h>
#include <owl/framewin.h>
```

```
#include <owl/decframe.h>
#include <owl/statusba.h>
#include <owl/controlb.h>
#include <owl/buttonga.h>
#include <owl/gdiobjec.h>
#include <owl/chooseco.h>
#include <owl/inputdia.h>
#include <owl/opensave.h>
#include <owl/dc.h>


#include <mem.h>

#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <stdio.h>
#include <stddef.h>
#include <string.h>

#include "tinwin2d.rh"

#define maxtriangle 20000
#define maxpoint 8000
#define maxarc 200

class TTINDrawApp : public TApplication
 {
  public:
    TTINDrawApp() : TApplication() {}  // call base class constructor
    void InitMainWindow();
 };

class TTINWindow : public TWindow
 {
  public:
    TTINWindow(TWindow* parent = 0)
      : TWindow(parent) {}

    struct Point
     {
      float x;
      float y;
      float z;
     };
```

```
struct Triangle
 {
  int Node1;
  int Node2;
  int Node3;
 };

Point* pnt_ptr[maxpoint];
Triangle* tri_ptr[maxtriangle];

struct Polygon
 {
  int Snode;
  int Enode;
 };

Polygon* poly_ptr[maxarc];

FILE* XYZfile;
FILE* TINfile;
FILE* ARCfile;


TOpenSaveDialog :: TData fileData;
int k, i, t, ii;
float xmin, xmax, ymin, ymax;
float zmin, zmax;
float xlength, ylength;
int npnt, ntri, narc;
int MaxX, MaxY;
float scaleX, scaleY;

void ReadXYZ();
void ReadTIN();
void ReadARC();
void GetMinMax(float&, float&,
            float&, float&,
            float&, float&);
void Paint(TDC& dc, bool, TRect&);
void EvSize(uint, TSize&);
void CmInputFilesItem1();
void CmPopupItem1();
void CmFileExit()
    {PostQuitMessage(0);}
void CmClear();
void CmAbout();
```

```
    DECLARE_RESPONSE_TABLE(TTINWindow);
};

DEFINE_RESPONSE_TABLE1(TTINWindow, TWindow)
 EV_WM_SIZE,
 EV_COMMAND(CM_INPUTFILESITEM1, CmInputFilesItem1),
 EV_COMMAND(CM_POPUPITEM1, CmPopupItem1),
 EV_COMMAND(CM_ABOUT, CmAbout),
 EV_COMMAND(CM_FILEEXIT, CmFileExit),
 EV_COMMAND(CM_CLEAR, CmClear),
END_RESPONSE_TABLE;
```

The 3D TIN Generation class:

```
//*************************************************************************//
//    Object-Oriented 3D Distance Transformation (DT) program for 3D TIN    //
//          Input  :  3D array format                                       //
//          Output :  3D array ILWIS look-like file format                  //
//                                                                          //
//*************************************************************************//

#include <iostream.h>
#include <stdlib.h>
#include <stdio.h>
#include <fstream.h>
#include <io.h>
#include <iomanip.h>
#include <dos.h>
#include <malloc.h>
#include <conio.h>
#include <limits.h>
#include <stddef.h>

// Constant section

#define masksize 27          // Mask size

class DistanceTransform3D
 {
  public:
     DistanceTransform3D();
```

```
 ~DistanceTransform3D();
// Data members
// VPI input file structure
typedef struct VpiStruct
        {
         short Nscanlines;      // no. of image rows
         short Npixels;         // no. of image cols
         short Nlevels;         // no. of image levels
         short Pvmin;           // voxel's minimum value
         short Pvmax;           // voxel's maximum value
         short Maptype;         // map type
         short Patch;           // patch type
         short Scale;           // image scale factor
         short Crdtype;         // coordinate type
         float a11;             // transformation cooefficient (3 x 3 matrix)
         float a12;             //              "
         float a13;             //              "
         float a21;             //              "
         float a22;             //              "
         float a23;             //              "
         float a31;             //              "
         float a32;             //              "
         float a33;             //              "
         float b1;              // translation vector (3 x 1 matrix)
         float b2;              //              "
         float b3;              //              "
        } VpiType;

VpiType VPI, VPIOut;

typedef short Image;
typedef Image* VoxelRow;
typedef VoxelRow* Voxel2D;
typedef Voxel2D* Voxel3D;

typedef short Mask[masksize];

// Global variables
Mask MaskPix;
FILE* VPDifile;
FILE* VPDofile;
FILE* VPIifile;


FILE* VPIofile;
```

```
    Voxel3D Voxel;          // variable for voxels
    int maxrow;
    int maxcol;

    int maxlevel;
    float scale;
    int row, col, level;
    int r, l;

    // Function members
    void GetVPDInputFile();
    void GetVPIInputFile(VpiType& VPI);
    void ReadVoxelImage(Voxel3D&);
    void PrintVoxel(Voxel3D);
    void WriteVPDOutputFile();
    void WriteVPIOutputFile();
    void SetBackground(Voxel3D, int, int);
    void GetUpperMask(int, int, int, Voxel3D, Mask&);
    void GetLowerMask(int, int, int, Voxel3D, Mask&);
    int MinByIndex(int, int);
    int Min5(int, int, int, int, int);
    void DistancePassOne(Voxel3D);
    void DistancePassTwo(Voxel3D);
    void ForwardDistance();
    void BackwardDistance();
    void DisplayVoxel(Voxel3D);
    void PressAnyKey();
    void Title();
};
```

The 3D Voronoi Tessellations class:

```
//*************************************************************************//
//        Object-Oriented 3D Distance Transformation (DT) and              //
//             Voronoi Tessellation program for 3D TIN                     //
//                                                                         //
//           Input  :  3D array format                                     //
//           Output :  3D array ILWIS look-like file format                //
//                        (i.e. .VPD, and .VPI)                            //
//                                                                         //
//*************************************************************************//
```

```
#include <iostream.h>
#include <stdlib.h>
#include <stdio.h>
#include <fstream.h>
#include <io.h>
#include <iomanip.h>
#include <dos.h>
#include <malloc.h>


#include <conio.h>
#include <limits.h>
#include <stddef.h>

#include "TDistanceTransform3D.h"

// Constant section

#define masksize 27              // Mask size

class TVoronoi3D : public TDistanceTransform3D
 {
  public:
    TVoronoi3D();          // constructor
    ~TVoronoi3D();         // destructor

   typedef short Mask[masksize];

   // Global variables
   Mask MaskPixVor;

   FILE* VPDifile;
   FILE* VPDodfile;
   FILE* VPDovfile;
   FILE* VPIifile;
   FILE* VPIodfile;
   FILE* VPIovfile;

   // variable definition
   Voxel3D Voxel, VoxelVor;          // variable for voxels

   // Functions Prototype;
   void GetVPDInputFile();
   void GetVPIInputFile(VpiType& VPI);
   void ReadVoxelImage(Voxel3D&);
```

```
    void CopyVoxel(Voxel3D, Voxel3D&);
    void PrintVoxel(Voxel3D);
    void WriteVPDOutputFileDist();
    void WriteVPIOutputFileDist();
    void WriteVPDOutputFileVoronoi();
    void WriteVPIOutputFileVoronoi();
    void SetBackground(Voxel3D, int, int);
    void GetUpperMaskDist(int, int, int, Voxel3D, Mask&);
    void GetLowerMaskDist(int, int, int, Voxel3D, Mask&);
    void GetUpperMaskVoronoi(int, int, int, Voxel3D, Mask&);
    void GetLowerMaskVoronoi(int, int, int, Voxel3D, Mask&);
    int MinByIndex(int, int);
    int Min5(int, int, int, int, int);
    void ForwardPass(Voxel3D, Voxel3D);
    void BackwardPass(Voxel3D, Voxel3D);
    void ForwardVoronoi();
    void BackwardVoronoi();
    void PressAnyKey();
    void Title();
  };
```

The 3D TIN View class:

```
//*********************************************************//
// A Program to View TIN with arcs and edges on screen     //
// A Program to Display 3D TINs Using OWL                  //
// Input    : (.XYZ), (.TIN), and (.ARC) files             //
// Output   : Windows display                              //
//                                                         //
//*********************************************************//

#include <owl/pch.h>
#include <owl/applicat.h>
#include <owl/framewin.h>
#include <owl/dc.h>
#include <owl/decframe.h>
#include <owl/statusba.h>
#include <owl/controlb.h>
#include <owl/buttonga.h>
#include <owl/gdiobjec.h>
```

```
#include <owl/chooseco.h>
#include <owl/inputdia.h>
#include <owl/opensave.h>
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <stdio.h>
#include <stddef.h>
#include <math.h>
#include <mem.h>
#include <string.h>

#include "view3d.rh"


// constants for maxpoints, tins, and arcs
const int maxtin = 2000;
const int maxpoint = 700;
const int maxarc = 200;
const int maxsegment = 1000;

// view parameters constants

const double theta = -60.0;  // was -60.0
const double phi = 60.0;
const double ViewPlaneDist = 2000.0; // 500 for the trylake
                             // the bigger the ViewPlaneDist
                             // -> larger view
const double rho = 2500.0;          // 1500 for the trylake

class TView3DWindowApp : public TApplication
 {
  public:


    TView3DWindowApp() : TApplication() {}  // call base class constructor
    void InitMainWindow();
 };


class TView3D : public TWindow
 {
  public:
    TView3D(TWindow* parent = 0)
          : TWindow(parent) {}
```

```
typedef double Matrix[4][4];
struct Vector
 {
  double  x, y, z;
 };

struct Point
 {
  double x, y, z;
 };

struct Pointt
 {
  float x, y, z;
 };

struct TIN
 {
  int Node1, Node2, Node3;
 };

struct TIN2
 {
  int Node1, Node2, Node3;
 };

Point* pnt[maxpoint];
Pointt* pntt[maxpoint];
TIN* tin[maxtin];
TIN2* tinn[maxtin];

struct Polygon
 {
  int Snode;
  int Enode;
 };
Polygon* poly_ptr[maxarc];

struct RoofArc
 {
  int Snode;
  int Enode;
 };
RoofArc* roofarc[maxarc];
```

```
FILE* ARCfile;
FILE* ARCRooffile;

// struct for trees
struct TreePoints
 {
  double x;
  double y;
  double z;
 };
TreePoints* treepnts[maxpoint];

struct TreeArcs
 {
  int Snode;
  int Enode;
 };
TreeArcs* treearc[maxarc];

// file for the tree points and arcs
FILE* Treepntfile;
FILE* Treearcfile;

struct Segment
 {
  double x;
  double y;
  int Hreq;
  int SegNr;
 };
Segment* Seg[maxsegment];

int i, t, tt, k;
int npnt, ntin, narc, n;
int npntt, ntinn, narcr;
int ntree, narct;
string charNode1, charNode2, charNode3;
double xmin, xmax, ymin, ymax;
double zmin, zmax;
double xlength, ylength, zlength;

double startx, starty;
double nextx, nexty;
int SegNr, nextSegNr, prevSegNr;
double x, y;
```

```
int h, nexth;
double MaxX, MaxY;
double xmid, ymid;
double screen_x, screen_y;
double scaleX, scaleY, scaleZ;
double sin_theta;
double cos_theta;
double sin_phi;
double cos_phi;
double x_view, y_view, z_view;


Matrix viewT;
Vector VecWorld, VecView;
double DotProduct(Vector, Vector);
double Magnitude(Vector);
void Transform(int, float&, float&);
void Normalize(Vector, Vector&);
void SetZeroMatrix(Matrix&);
void SetIdentityMatrix(Matrix&);
void VectorMatrix(Vector, Matrix, Vector&);
void SetViewTransformationMatrix(Matrix);
void SetViewVariables(double&, double&, double&);
void ReadXYZ();
void ReadXYZ2();
void ReadTIN();
void ReadTIN2();
void ReadARC();
void ReadRoofArc();
void ReadTreeXYZ();
void ReadTreeArc();
void ReadSegment();
void DeallocateMemory();
void GetMinMax(double&, double&, double&, double&, double&, double&,
        double&, double&, double&);
void Perspective(double, double, double, double&, double&);
double InRadians(double);
void Paint(TDC& dc, bool, TRect&);
void EvSize(uint, TSize&);
void CmInputFilesItem1();
void CmPopupItem1();
void CmFileExit()
   {PostQuitMessage(0);}
void CmClear();
```

```
    void CmAbout();

    ~TView3D() {}

    DECLARE_RESPONSE_TABLE(TView3D);
  };

DEFINE_RESPONSE_TABLE1(TView3D, TWindow)
  EV_WM_SIZE,
  EV_COMMAND(CM_INPUTFILESITEM1, CmInputFilesItem1),
  EV_COMMAND(CM_POPUPITEM1, CmPopupItem1),
  EV_COMMAND(CM_ABOUT, CmAbout),
  EV_COMMAND(CM_FILEEXIT, CmFileExit),
  EV_COMMAND(CM_CLEAR, CmClear),
END_RESPONSE_TABLE;
```

Appendix C

Class Definitions for POET Database Schema

The following are definitions of the classes implemented under POET OO DBMS software (header files which are compatible with the POET environment).

The TNode class:

```
//************************************//
// The database with POET             //
// Alias Abdul-Rahman (c) March, 2000  //
//                                    //
//************************************//

//#include <iostream.h>
//#include <stdio.h>
//#include <stdlib.h>
#include "ptstring.hxx"

#include "xyzcontainer.hcd"

class NodeAtrContainer
{
public:
  int NodeNum;
  PtString NodeName;
};

#define maxpoint 10000
#define maxnodename 100

persistent class TNode
  {
   public:
     XYZContainer Point[maxpoint];
     NodeAtrContainer NodeAtr[maxnodename];
     int npnt;
     TNode();
     ~TNode();

     void GetXYZCoordinates();
```

```
    void GetBoreholeCoordinates();
    void Get2Node();
    void NodeAttribute();
    void PressAnyKey();

  };
```

The TEdge class:

```
//************************************//
// The database with POET             //
// Alias Abdul-Rahman (c) March, 2000  //
//                                    //
//************************************//

#include "ptstring.hxx"
#include "arccontainer.hcd"
#include "xyzcontainer.hcd"

#define maxarc 500
#define maxarcname 100

persistent class TNode;
persistent class TEdge
{
public:
    friend persistent class TNode;
    int narc;

   ARCContainer Arc[maxarc];
   ARCAtrContainer EdgeAtr[maxarcname];

    TEdge();
    ~TEdge();

    void ReadARCs();
    void GetOneArc();
    void GetArcLength();
    void GetArcAttribute();
    int CheckQuadrant(float, float);
    float Bearing(float, float, float, float);
```

```
    float GetArcAzimuth(float, float, float, float);
    void EdgeAttribute();
    void PressAnyKey();
    friend XYZContainer;
};

class PtOnDemand;
typedef lset<ondemand<TEdge>> TedgeSet;
```

The TPoly class:

```
//*************************************//
// The database with POET              //
// Alias Abdul-Rahman (c) March, 2000  //
//                                     //
//*************************************//

#include "tincontainer.hcd"
#define maxtriangle 1000

persistent class TNode;
persistent class TEdge;
persistent class TPoly
 {
  public:
    double xNd1, yNd1, zNd1;
    double xNd2, yNd2, zNd2;
    double xNd3, yNd3, zNd3;

    friend persistent class TNode;
    friend persistent class TEdge;

    TINContainer Triangle[maxtriangle];
    TENPolyContainer TINNbr[maxtriangle];

    TPoly();
    ~TPoly();

    void ReadTINs();
    void GetTINNeighbour();
    void GetTINNodes(int,
                double&, double&, double&,
```

```
                double&, double&, double&,
                double&, double&, double&);
    float GetTINArea(double, double,
                double, double,
                double, double);
    void GetPolyArea();
    void PressAnyKey();
  };
```

The TSolid class:

```
//************************************//
// The database with POET             //
// Alias Abdul-Rahman (c) March, 2000  //
//                                    //
//************************************//

#include "tincontainer.hcd"
#include "xyzcontainer.hcd"


#define maxtriangle 10000

persistent class TNode;
persistent class TSolid
 {
  public:
    double xNd1, yNd1, zNd1;
    double xNd2, yNd2, zNd2;
    double xNd3, yNd3, zNd3;
    double xNd4, yNd4, zNd4;

    TENContainer TEN[maxtriangle];

    TSolid();
    ~TSolid();

    void ReadTENs();
    void Get3TINNodes(int,
                double&, double&, double&,
                double&, double&, double&,
                double&, double&, double&,
                double&, double&, double&);
```

```
    double ComputeTENVolume(double, double, double,
                            double, double, double,
                            double, double, double,
                            double, double, double);

    void GetVolume();
    friend persistent class TNode;
    void PressAnyKey();
};
```

The TIN Neighbour class:

```
class Neighbour
 {
  public:
    int TriNum;

    int TotalNeighbour;
    int Nbr[3];
    Neighbour()
     {
      TriNum = 0;
      TotalNeighbour = 0;
      Nbr[0] = Nbr[1] = Nbr[2] = 0;
     };
    ~Neighbour(){};
 };


class TinNeighbour
 {
  public:
     // structure for the TINs
     struct ThreeNodes
      {
       int Node[3];
      };
     ThreeNodes* tri[maxtriangle];

    // some global variables
    FILE* TINfile;
```

```
    FILE* NBRfile;
    int t;
    int ntri;
    bool isTriangleNode;

    // some operations
    TinNeighbour();
    ~TinNeighbour();

    void GetTINfile();
    void GetNeighbourfile();
    bool Less(int a, int b);
    bool Greater(int, int);
    void Swap(int&, int&);
    void NodeOrder(int&, int&, int&);
    void AddNbrtoFile(Neighbour);
    void MakeTinNeighbour();
    void PressAnyKey();
};
```

File Formats

The following gives detailed descriptions of the file formats developed in this work. Three formats were adopted from commercial software, namely Arc/Info, ILWIS and AVS visualization software.

These are the physical structures of the files. The files have the following three characters (suffix):

TRI, NBR, XYZ, TRS, SID, SEG, and LIN formats.

The commercial formats are:

.MPD, .MPI file format of the ILWIS package, and also the volume data file format of the AVS package (i.e., the .DAT file).

- The TRI file format (triangles' three-nodes):

```
Triangle#     Node1   Node2   Node3
..      ...             ...             ...
EOF
```

- The NBR file format (triangles' neighbours):

```
Triangle#     Number_of_Neighbour   Neighbour1     Neighbour2     Neighbour3
..      ...             ...             ...             ...             ...
EOF
```

- The TRS file format (triangles' edge and the right and left triangles):

```
Sides#          Node1   Node2   RightTriangle     LeftTriangle

..      ...      ...             ...               ...

EOF
```

- The SID file format (triangles' three sides):

```
Triangle#      Side1    Side2    Side3

..      ...      ...        ...

EOF
```

- The SEG file format (contour segments):

```
X  Y  ContourHeights Segment#

..  ...      ...                  ....

EOF
```

- The XYZ file format (coordinates):

```
X  Y  Z

..  ...…

EOF
```

- The LIN Arc/Info file format:

```
Code
X Y
..
..
END
..
..
END
END
```

- The MPI ILWIS file format:

  File with raster map information. It occupies 40 bytes of disk space, and containing the following information:

1..2 bytes	number of lines in map (integer)
3..4 bytes	number of columns of in map (integer)
5..6 bytes	minimum value in map (integer)
7..8 bytes	maximum value in map (integer)
9..10 bytes	map type (integer)
	0: bit map; 1: bytes map; 2: integer map
11..12 bytes	Patch map indication (integer)
	0: no patch map; 1: patch map
13..14 bytes	scale power (integer)
	scale factor represents 10 power multiplication factor
15..16 bytes	coordinate system type (integer)
	0: coordinates; 1: metric; 2: geographic
17..32 bytes	Multiplication matrix (2 x 2 float numbers)
33..40 bytes	Translation vector (2 x float numbers)

- The MPD ILWIS file format

  File with raster map data. The data is stored line by line, pixel by pixel.

  For example for a byte map with 480 lines and 640 column:

- 1st. pixel of the 1st. line
- 2nd. pixel of the 1st. line

...

- 640th. pixel of the 1st. line
- 1st. pixel of the 2nd. line

...

...

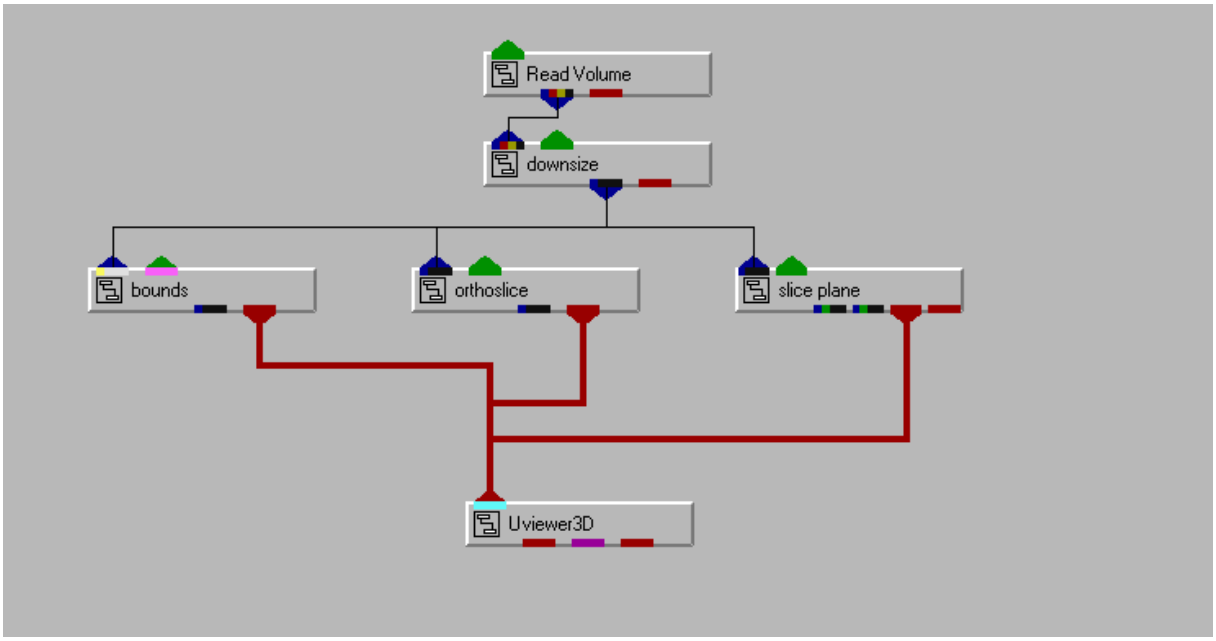
307200: 640th. pixel of the 480th. line

- The DAT file format of the Advance Visualization System (AVS)

  Contains the file header of number of row, columns and the level (3rd. dimension) of the data sets. Then followed by the actual data sets for each row, column, and level. It is a binary file.

The 3D Raster Image Visualization via AVS™

The following gives the general flow for displaying the 3D raster images (i.e. DT images and Voronoi tessellations images) in the AVS Visualization/Express software.



The figure shows six rectangles of operations (i.e. input file to images display). The ReadVolume is for inputting the 3D raster files (.DAT files). The downsize rectangle is for reducing the size of the raster file (normally to speed up the process). The bound is for generating the boundary of the view. The orthoslice and slice plane are for the cross section views. The Uviewer3D is for the general 3D display on screen.

TEN-based Volume

This type of volume is particularly useful because it is often possible to split more complicated solids into tetrahedra and to add up the volumes of those tetrahedra to find the total volume of the solid. The volume of a tetrahedron is found by forming a determinant from the three vectors obtained by subtracting one vertex (or node) from other three and dividing by 6 (Bowyer and Woodwark, 1983).

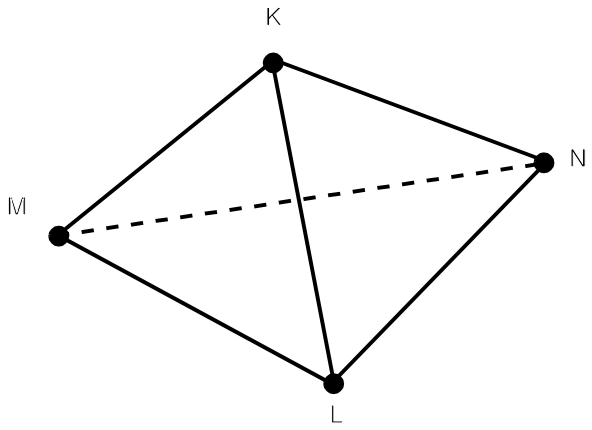


Figure F.1 The four nodes of a tetrahedron

From the four nodes (each has X, Y, Z coordinates) of a tetrahedron, it volume can be formulated as follows.

$$\frac{1}{6}\begin{vmatrix}(X_L - X_K)(X_M - X_K)(X_N - X_K)\\(Y_L - Y_K)(Y_M - Y_K)(Y_N - Y_K)\\(Z_L - Z_K)(Z_M - Z_K)(Z_N - Z_K)\end{vmatrix}$$

Class Definitions for MDI Windows Interface

The Main Application class:

```
//****************************************************************************//
//  Project: MDI for 2D/3D TINs                                               //
//                                                                            //
//                                                                            //
//  SUBSYSTEM:    Graphing Application                                        //
//  FILE:         graphingapp.h                                               //
//                                                                            //
//  OVERVIEW                                                                  //
//  ~~~~~~~~                                                                  //
//  Class definition for GraphingApp (TApplication).                          //
//                                                                            //
//  Author: Alias Abdul-Rahman (c) 1999                                       //
//****************************************************************************//

#if !defined(graphingapp_h)        // Sentry, use file only if it's not
#define graphingapp_h              //   already included.

#include <owl/controlb.h>
#include <owl/docking.h>
#include <owl/printer.h>
#include <owl/rcntfile.h>

#include "graphingmdiclient.h"

#include "graphingapp.rh"          // Definition of all resources.


//{{TApplication = GraphingApp}}
class GraphingApp : public TApplication, public TRecentFiles {
  private:
    void SetupSpeedBar(TDecoratedMDIFrame* frame);

  public:
    GraphingApp();
    virtual ~GraphingApp();
```

```
    void CreateGadgets(TDockableControlBar* cb, bool server = false);
    THarbor*        ApxHarbor;

    TGraphingMDIClient* MdiClient;

    // Public data members used by the print menu commands
    // and Paint routine in MDIChild.
    //


    TPrinter*       Printer;              // Printer support.
    int             Printing;             // Printing in progress.


    public:
    virtual void InitMainWindow();

  protected:
    void EvNewView(TView& view);
    void EvCloseView(TView& view);
    void CmHelpAbout();
    void EvWinIniChange(char far* section);
    void EvOwlDocument(TDocument& doc);
    int32 CmFileSelected(uint wp, int32 lp);

DECLARE_RESPONSE_TABLE(GraphingApp);
};    //{{GraphingApp}}

#endif  // graphingapp_h sentry.
```

The Document class:

```
//******************************************************************************//
//  Project: MDI for 2D/3D TINs                                                 //
//                                                                              //
//                                                                              //
//  SUBSYSTEM:    graphing.apx Application                                      //
//  FILE:         graphdocument.h                                               //
//                                                                              //
//  OVERVIEW                                                                    //
//  ~~~~~~~~                                                                    //
//  Class definition for GraphDocument (TFileDocument).                         //
//                                                                              //
```

```
//  Author: Alias Abdul-Rahman (c) 1999                                  //
//***************************************************************************//

#if !defined(graphdocument_h)  // Sentry, use file only if it's not already
#define graphdocument_h        //    included.

#include <owl/filedoc.h>

#include "graphingapp.rh"            // Definition of all resources.
#include <list>
#include <vector>
    struct Point
     {
      float x;
      float y;
      float z;
      int operator ==(const Point& p)
         const
           {
            return x == p.x &&
                   y == p.y &&
                   z == p.z;
           }

      int operator <(const Point& p)
         const
          {
           return x < p.x &&
                  y < p.y &&
                  z < p.z;
          }
     };

    struct Triangle
     {
      int node_1;
      int node_2;
      int node_3;
      int operator ==(const Triangle& t)
         const
          {
           return node_1 == t.node_1 &&
                  node_2 == t.node_2 &&
```

```
                node_3 == t.node_3;
            }

        int operator <(const Triangle& t)
            const
            {
             return node_1 < t.node_1 &&
                    node_2 < t.node_2 &&
                    node_3 < t.node_3;
            }
       };


      struct Segment
        {
         float x;
         float y;
         int Hreq;
         int SegNr;
         int operator ==(const Segment& seg)
            const
             {
              return x     == seg.x    &&
                     y     == seg.y    &&
                     Hreq  == seg.Hreq &&
                     SegNr == seg.SegNr;
             }

         int operator <(const Segment& seg)
            const
             {
              return x     < seg.x    &&
                     y     < seg.y    &&
                     Hreq  < seg.Hreq &&
                     SegNr < seg.SegNr;
             }
       };

class GraphDocument : public TFileDocument {
  public:
    GraphDocument(TDocument* parent = 0);
    virtual ~GraphDocument();
public:
    void GetXYZ_TIN()
```

```
    {
     if (!IsOpen())
       Open(ofRead) ;
    }


   virtual bool Open(int mode, const char far* path=0);

// Use STL containers to make life easier for
//      points, triangles, contour segments, etc.
  std::vector<Point> points;
  std::list<Triangle> triangles;
  std::list<Segment> contours;

  float xmin, ymin, xmax, ymax, zmin, zmax;
  };    //{{GraphDocument}}

#endif  // graphdocument_h sentry.
```

The View class:

```
//****************************************************************************//
//  Project: MDI for 2D/3D TINs                                               //
//                                                                            //
//                                                                            //
//  SUBSYSTEM:    Graphing Application                                        //
//  FILE:         graphingwindowview.h                                        //
//                                                                            //
//  OVERVIEW                                                                  //
//  ~~~~~~~~                                                                  //
//  Class definition for TGraphingWindowView (TWindowView).                   //
//                                                                            //
//  Author: Alias Abdul-Rahman (c) 1999                                       //
//****************************************************************************//

#if !defined(graphingwindowview_h)   // Sentry, use file only if it's not
#define graphingwindowview_h         // ... already included.
#include <owl/docview.h>
#include "graphingapp.rh"            // Definition of all resources.

class TGraphingWindowView : public TWindowView {
  public:
    TGraphingWindowView(TDocument& doc, TWindow* parent = 0);
    virtual ~TGraphingWindowView();
```

```
  public:
    virtual void Paint(TDC& dc, bool erase, TRect& rect);

  protected:
    void EvGetMinMaxInfo(MINMAXINFO far& minmaxinfo);


    void EvSize(uint sizeType, TSize& size);

DECLARE_RESPONSE_TABLE(TGraphingWindowView);
};    //{{TGraphingWindowView}}

#endif  // graphingwindowview_h sentry.
```

The Client class:

```
//******************************************************************************//
//  Project: MDI for 2D/3D TINs                                                 //
//                                                                              //
//                                                                              //
//  SUBSYSTEM:    Graphing Application                                          //
//  FILE:         graphingmdiclient.h                                           //
//                                                                              //
//  OVERVIEW                                                                    //
//  ~~~~~~~~                                                                    //
//  Class definition for TGraphingMDIClient (TMDIClient).                       //
//                                                                              //
//  Author: Alias Abdul-Rahman (c) 1999                                         //
//******************************************************************************//

#if !defined(graphingmdiclient_h)    // Sentry, use file only if it's not
#define graphingmdiclient_h          // already included
#include "graphingapp.rh"            // Definition of all resources.

class TGraphingMDIClient : public TMDIClient {
  public:
    int      ChildCount;             // Number of child window created.
    TGraphingMDIClient(TModule* module = 0);
    virtual ~TGraphingMDIClient();

    void OpenFile(const char* fileName = 0);

  protected:
    virtual void SetupWindow();
```

```
  protected:
    void CmFilePrint();
    void CmFilePrintSetup();
    void CmFilePrintPreview();
    void CmPrintEnable(TCommandEnabler& tce);

DECLARE_RESPONSE_TABLE(TGraphingMDIClient);
};    //{{TGraphingMDIClient}}


#endif  // graphingmdiclient_h sentry.
```

The Child class:

```
//******************************************************************************//
//  Project: MDI for 2D/3D TINs                                                 //
//                                                                              //
//                                                                              //
//  SUBSYSTEM:    Graphing Application                                          //
//  FILE:         graphingmdichild.h                                            //
//                                                                              //
//  OVERVIEW                                                                    //
//  ~~~~~~~~                                                                    //
//  Class definition for TGraphingMDIChild (TMDIChild).                         //
//                                                                              //
//  Author : Alias Abdul-Rahman  (c) 1999                                       //
//******************************************************************************//

#if !defined(graphingmdichild_h)     // Sentry, use file only if it's not
#define graphingmdichild_h           // not alraedy included
#include "graphingapp.rh"            // Definition of all resources.

class TGraphingMDIChild : public TMDIChild {
  public:
    TGraphingMDIChild(TMDIClient& parent, const char far* title,
                      TWindow* clientWnd, bool shrinkToClient = false,
                      TModule* module = 0);
    virtual ~TGraphingMDIChild();
};    //{{TGraphingMDIChild}}

#endif  // graphingmdichild_h sentry.
```

The AboutDialog class:

```
//****************************************************************************//
//  Project: MDI for 2D/3D TINs                                               //
//                                                                            //
//                                                                            //
//  SUBSYSTEM:    Graphing Application                                        //
//  FILE:         graphingaboutdlg.h                                          //
//                                                                            //
//  OVERVIEW                                                                  //
//  ~~~~~~~~                                                                  //
//  Class definition for GraphingAboutDlg (TDialog).                          //
//                                                                            //
//  Author: Alias Abdul-Rahman (c) 1999                                       //
//****************************************************************************//

#if !defined(graphingaboutdlg_h)   // Sentry, use file only if it's not
#define graphingaboutdlg_h         // already included.
#include <owl/static.h>
#include "graphingapp.rh"          // Definition of all resources.

class GraphingAboutDlg : public TDialog {
  public:
    GraphingAboutDlg(TWindow* parent, TResId resId = IDD_ABOUT,
                     TModule* module = 0);
    virtual ~GraphingAboutDlg();

  public:
    void SetupWindow();
};    //{{GraphingAboutDlg}}

class TProjectRCVersion {
  public:
    TProjectRCVersion(TModule* module);
    virtual ~TProjectRCVersion();
    bool GetProductName(LPSTR& prodName);
    bool GetProductVersion(LPSTR& prodVersion);
    bool GetCopyright(LPSTR& copyright);
    bool GetDebug(LPSTR& debug);

  protected:
    uint8 far*  TransBlock;
```

```
    void far*   FVData;

  private:
    TProjectRCVersion(const TProjectRCVersion&);
    TProjectRCVersion& operator = (const TprojectRCVersion&);
};

#endif  // graphingaboutdlg_h sentry.
```

Curriculum Vitae

The following gives a brief curriculum vitae of the author of this thesis, *Alias Abdul-Rahman*. He was born in Malaysia in January 1962. Received education in Malaysia, England, The Netherlands, and currently in Scotland, U.K.

Education

- Diploma in Land Surveying (UTM), Malaysia (1980-1983).
- B.Sc (Hons.) in Surveying and Mapping Sciences, Univ. of East London, formerly known North East London Polytechnic (NELP) (1984-1987).
- Post. Grad. Dipl. in Integrated Map and Geoinformation Production (ITC, The Netherlands) (1990-1991).
- M.Sc in Integrated Map and Geoinformation Production (ITC, The Netherlands) (1991-1992).
- PhD in GIS (University of Glasgow) (October1996 -September 2000).

Employment

- Assistant Lecturer at Universiti Teknologi Malaysia (UTM) (1987-1990)
- Lecturer at UTM (1992 till present).
- Head, Department of Geoinformatics (UTM) (1993-1995).

Computing Skills

Highly competent in Object-Oriented (OO) programming particularly C++ as well as structural programming with Turbo Pascal. Also competent in Windows-based programming with Object Windows Library (OWL).

Awards

Excellent Academic Staff Award, UTM (1995).

Publication and Conference Presentations

ABDUL-RAHMAN, A., and DRUMMOND, J. (2000). The Implementation of Object-Oriented TIN-based Subsystems for GIS. *International Society of Photogrammetry and Remote Sensing (ISPRS) Congress*, Amsterdam, The Netherlands, July, Commission 4, CD 1.

ABDUL-RAHMAN, A. (1999). Spatial tessellations using an object-oriented approach. *Proceedings of GIS Research UK 7th. Annual Conference (GISRUK '99)*, Southampton, U.K , 14-16th. April, pp. 13-23.

ABDUL-RAHMAN, A., and DRUMMOND, J. (1999). The development of 2D and 3D triangular irregular networks computer programs. *Proceedings of American Congress on Surveying and Mapping (ACSM) Conference and Exhibition,* Portland, Oregon, USA. 13-17 March, pp. 38-49.

ABDUL-RAHMAN, A., and DRUMMOND, J. (1998). Raster-based algorithms for 2D and 3D TINs generations. *Proceedings of International Conference of Spatial Information Science and Technology (SIST' 98)*, December, Wuhan, China.

ABDUL-RAHMAN, A., DRUMMOND, J., and SHEARER, J. W. (1998). Representation of 3D Spatial Objects, *American Congress on Surveying and Mapping (ACSM) Annual Conference and Exhibition*, 2-4 March, Baltimore, USA.

ABDUL-RAHMAN, A., and DESA, G. (1996). Identification of Developable Land Using TIN-based Digital Terrain Modelling, *International Archives of Photogrammetry and Remote Sensing*, Vol. XXXI, Part B4, Commission 4, Vienna, Austria, pp. 7-11.

ABDUL-RAHMAN, A. (1995). Digital Terrain Modelling Using Contour and GPS Data, *Proceedings of the 1st. Joint European Conference and Exhibition on Geographic Information*, Den Haag, The Netherlands, Vol. 2, pp. 216-218.

ABDUL-RAHMAN, A. (1994). Design and Evaluation of TIN Interpolation Algorithms, *Proccedings of the 5th. European Conference and Exhibition on Geographic Information Systems*, Paris, France, Vol.1, pp. 328-343.

ABDUL-RAHMAN, A., KUNARAK, and ROKHYADI. (1993). Digital Topographic Database for GIS and Digital Mapping: A Case Study of Luberon National Park, Southern France, *Proceedings of 13th. Annual ESRI User Conference*, Palm Springs, California, USA. Vol. 1, pp. 243-260.

Appendix I

Additional Papers

Two papers are presented in the following pages. The first paper was presented at the American Congress of Surveying and Mapping (ACSM) Conference and Exhibition, March, 1999, Portland, Oregon, USA. The second paper is for the International Society of Photogrammetry and Remote Sensing (ISPRS) Congress, Amsterdam, The Netherlands, July 2000. This particular paper has been published in the ISPRS Archive 2000 for Commission 4, CD 1.

The Development of 2D and 3D Triangular Irregular Network Programs

Alias Abdul-Rahman and **Jane E. Drummond**
Department of Geography and Topographic Science
University of Glasgow
Glasgow G12 8QQ
United Kingdom

Email:
alias@geog.gla.ac.uk
jdrummond@geog.gla.ac.uk

Abstract
This paper concentrates on a development of two-dimensional (2D) and three-dimensional (3D) Triangular Irregular Network (TIN) in raster domain. Two programs for generating the 2D TIN and 3D TIN were presented. Fundamental concepts involved in the program development, i.e. the concept of Borgefors Distance Transformation (DT), and Voronoi tessellations were highlighted. We demonstrated the application of the algorithms using rasterized data sets generated by our 2D and 3D rasterizing programs. Examples of the 2D and 3D TIN were given. Finally, the paper discusses the possibility of using the generated TIN data structures for future development of 3D spatial information system.

1.0 Introduction
Data structuring for terrain surface data has been studied and investigated since the late seventies (Peucker *et al.*,1978), when the suitability and the adaptability of data structures for terrain surface representation were considered. Then a triangular irregular network (TIN) data structure was established. Several methods and techniques have emerged for TIN generation since then (McCullagh and Ross, 1980; Watson, 1981; Mirante and Weingarten, 1982). Most of the developments were in the vector-based domain and computationally complex (de Berg, 1997). An alternative to the vector-based algorithm, a raster approach, was proposed (Pilouk, 1992; Chen *et al*, 1994; Pilouk, 1996). Since the alternative approach provides for far less complex computation, we adapted this approach in our 2D and 3D TIN development with an intention of developing a 3D spatial information system. The 2D and 3D TIN are part of the ongoing research and they form the major discussion of this paper.

We will describe the development of the 2D TIN program in the second section, and the 3D TIN program follows in the third section. The programs involve three major tasks; distance transformation, Voronoi or Dirichlet tessellations, and triangulations. The end result of each development is presented graphically in the form of a wireframe visualization. The remainder of the paper is devoted to the discussion on the possibility of further development of the TINs programs in object-oriented (OO) environment with a 3D GIS prototype in mind.

2.0 The 2D TIN Program

The program consists of three tasks. The tasks are:

- Distance transformation,
- Voronoi tessellation, and
- Triangulation.

Distance transformation (DT)

The task of DT is to generate a distance-transformed image of object pixels. Object pixels of a raster image may be in the form of random points, digitized points, digitized lines, etc. In this algorithm, the DT works as follows: all object pixel's are changed to zero (i.e. a value 0) and the rest of the pixels (i.e. the background pixels) to the highest possible value. In this case, the highest integer value of 32767 could be used. Second, scan the image in two passes (i.e. forward and backward passes) using a 3 x 3 mask of Chamfer 3-4 of the Borgefors DT (Borgefors, 1986), see Figure 1.

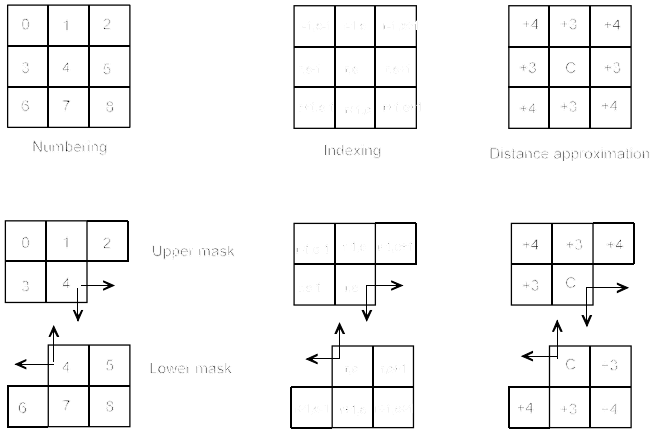


Figure 1 Masks for the DT

The forward pass (scans with upper-mask) begins from the first or top-left pixel and goes to the last pixel of the image. During the forward pass, all the pixels which were covered by the mask get a new value. Each pixel's value was added either to 3 or 4 depending on the pixel location. Then, the minimum value is determined from the five possible candidates and assigned to the current pixel location. The mask is then moved to the next pixel location. At this next location, the minimum value for this pixel is again determined and assigned. This process continues to the last pixel location (i.e. the bottom-right pixel) in the image. The result of the forward operation is used for the second pass which operates in reverse (from the last pixel to the first pixel). It is a recursive operation. Finally, a DT image is created after these two passes were carried out. Thus, all pixels will contain the approximate distance to the nearest object pixel.

Voronoi tessellation

The Borgefors DT and Voronoi tessellation tasks could be performed in parallel or in stages. In this algorithm, we carried out the tasks in parallel. If we re-examined the DT as described in the previous section, it involved three steps, first, change the object pixel value to zero (i.e. 0) and the background image to the highest possible value. Second, determine the minimum value of the current pixel location among five possible candidates of the upper mask. Third, assign the minimum value to the current pixel location. In other words, the pixel value represents a distance value of the pixel as calculated from the nearby object pixels. To obtain the Voronoi-tessellated image in parallel, the value of an original pixel needs to be assigned to the current pixel location

and written to a different output file. Figure 2 shows the parallel process for the DT and Voronoi tessellation.

In the Voronoi tessellation task, the result of the forward pass is used as an input for the backward pass. We tested the algorithm by using digitized contours. Figure 3 and Figure 4 show the DT image and the corresponding Voronoi tessellation image.

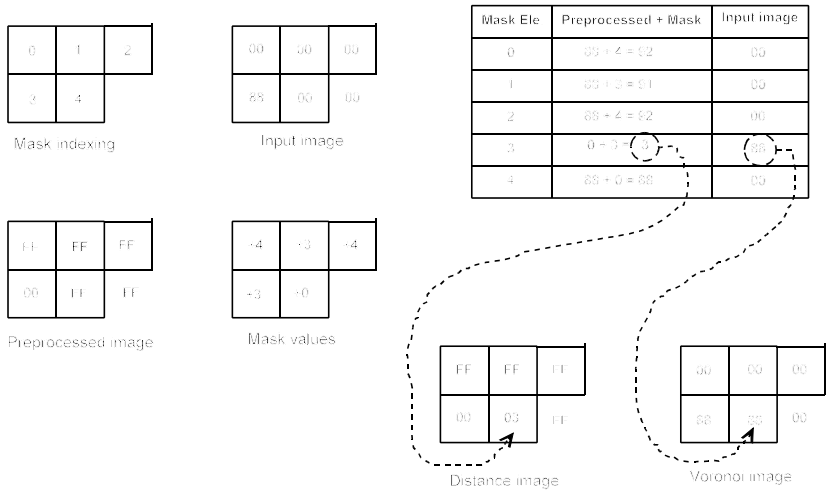


Figure 2 DT computation and Voronoi image generation during the forward pass.

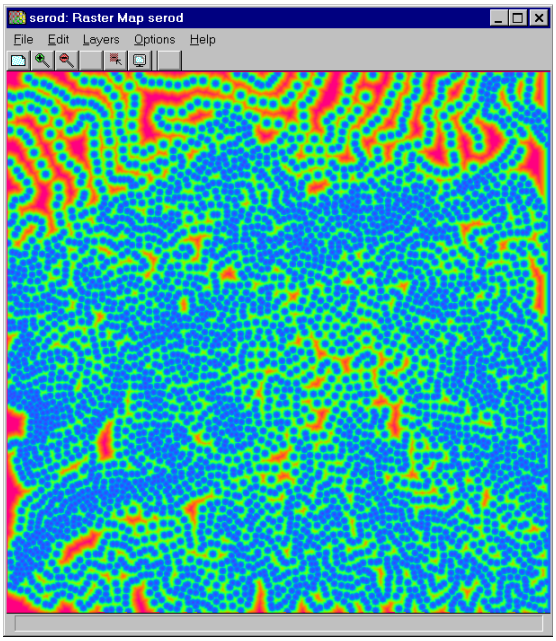


Figure 3 DT image of digitized points

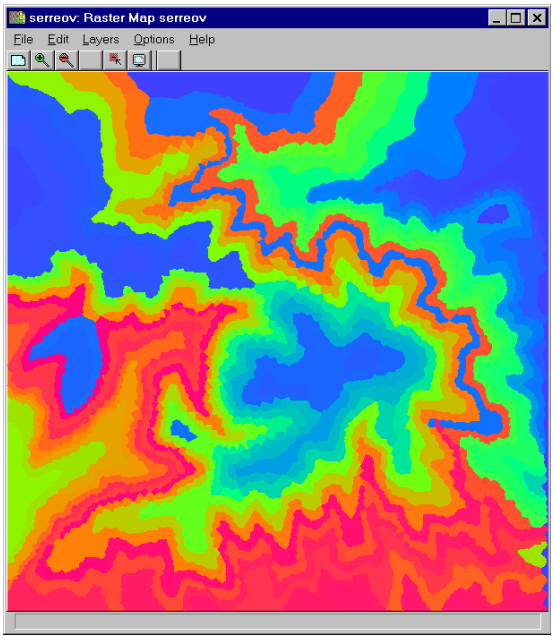


Figure 4 The corresponding Voronoi image of the digitized points

2D Triangulation

In this development, triangles were generated based on the Delaunay triangulation principle and attributed to Delaunay (1934). As shown in Figure 5, basically we need three kernel points of the neighbouring Voronoi polygons to form a triangle. If we have more than three neighbouring polygons, say 4 polygons, then there will be two possibilities for triangles formation.

A correct TIN topology must be established, and this is done by searching 3 Voronoi polygon neighbours. In order to find a unique set of 3 points from a Voronoi-tessellated image, a 2 x 2 mask is used. The mask is designed to detect only two specific situations where 3 or 4 different pixel values fall inside the mask at a time. These different pixels correspond to the neighbouring Voronoi polygons and the kernel points of these polygons were used to form the triangle. Figure 6 shows the mask for detecting the triangle topology.

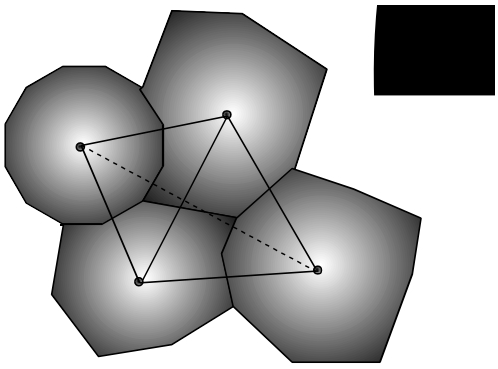


Figure 5 The two possible triangles formation

A triangle is found if the four different pixels match either of the imposed matching conditions. The mask was separated into two parts with the aim of avoiding overlapping (crossover) triangles, a situation not allowed in the Delaunay triangulation criteria.

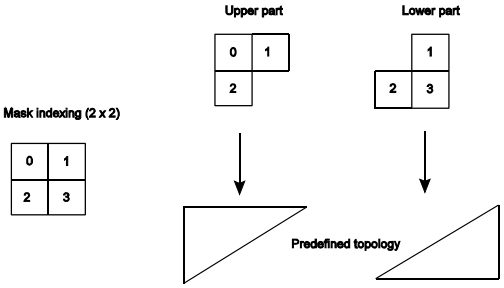


Figure 6 Mask (2 x 2) for TIN topology detection

The next task in the pipeline is to establish the triangle's attributes so that one could make use of the generated triangles, e.g. for queries and visualization purposes. For query purposes, a more complicated structure needs to be established. Some of the issues in TIN data structuring were studied by Abdul-Rahman (1992). However, in this development, a simple structure is devised where the purpose is only for visualizing the TINs; see Figure 7 for the structure.

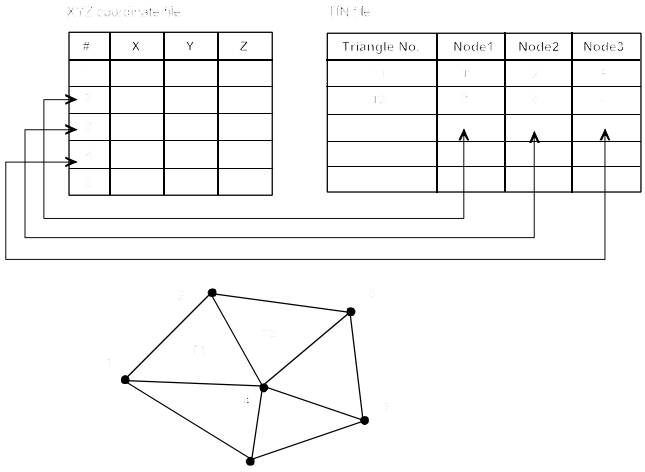


Figure 7 The link of XYZ coordinates and the TINs

TINs visualization

It has been claimed in de Berg (1997) that visualization of TINs is one of the major issues in TIN development. In this paper we have only developed a simple display program for visualizing generated TINs. One of the fundamental tasks of any GIS or DTM package is to perform the visualization of data.

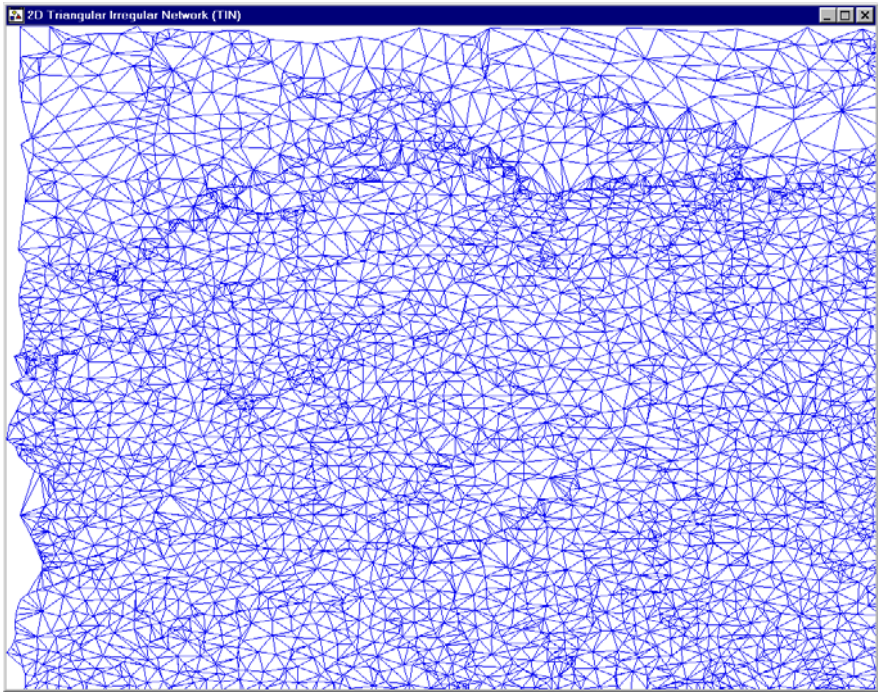


Figure 8 2D TIN visualization

Figure 8 shows a simple TIN visualization output developed in this process. The program takes two input files, a XYZ coordinate file, and the TIN table file. The triangles three nodes (i.e. Node1, Node2, and Node3) can be linked to the corresponding XYZ coordinate table for the nodes with the appropriate pointers as shown in Figure 7. Based on values in the XYZ file triangles could be shaded according to slope, elevation etc., for further visualization.

3.0 The 3D TIN Program

3D Distance transformation

Digital distance transformations in 3D have been considered for more than a decade and the technique has been applied in a number of scientific fields such as image processing (Borgefors, 1996). In this paper, we utilised the DT technique to generate a DT image, a Voronoi image, and tetrahedrons. The previous 2D DT algorithm can be extended to the third dimension relatively straightforwardly by due to the nature of the raster data structure. Thus, the same DT principle is utilised for the 3D TIN development. We used a 3D mask of dimension 3 x 3 x 3 as proposed by Borgefors (1996) known as Chamfer 3-4-5, see Figure 9. Other types of mask are also applicable such as the Chessboard mask, and the City-block mask (Borgefors, 1996).

The Chamfer mask is used due to its computational simplicity and is capable of generating quite accurate distance images. Each voxel in the mask is assigned a local distance either with a value 3, 4 or 5, depending on the voxel location, see Figure 9. The centre voxel is surrounded by 26 other voxels in x, y, z directions, where each voxel has three types of neighbours. They are called face neighbours, edge neighbours, and node or vertex neighbours. The face neighbour voxels are assigned the value 3, the edge voxels the value 4, and the vertex voxels the value 5.

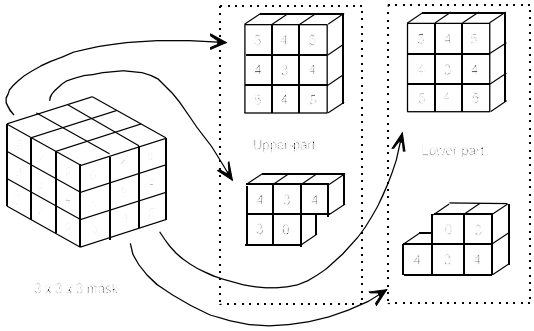


Figure 9 The 3-4-5 mask for the 3D DT

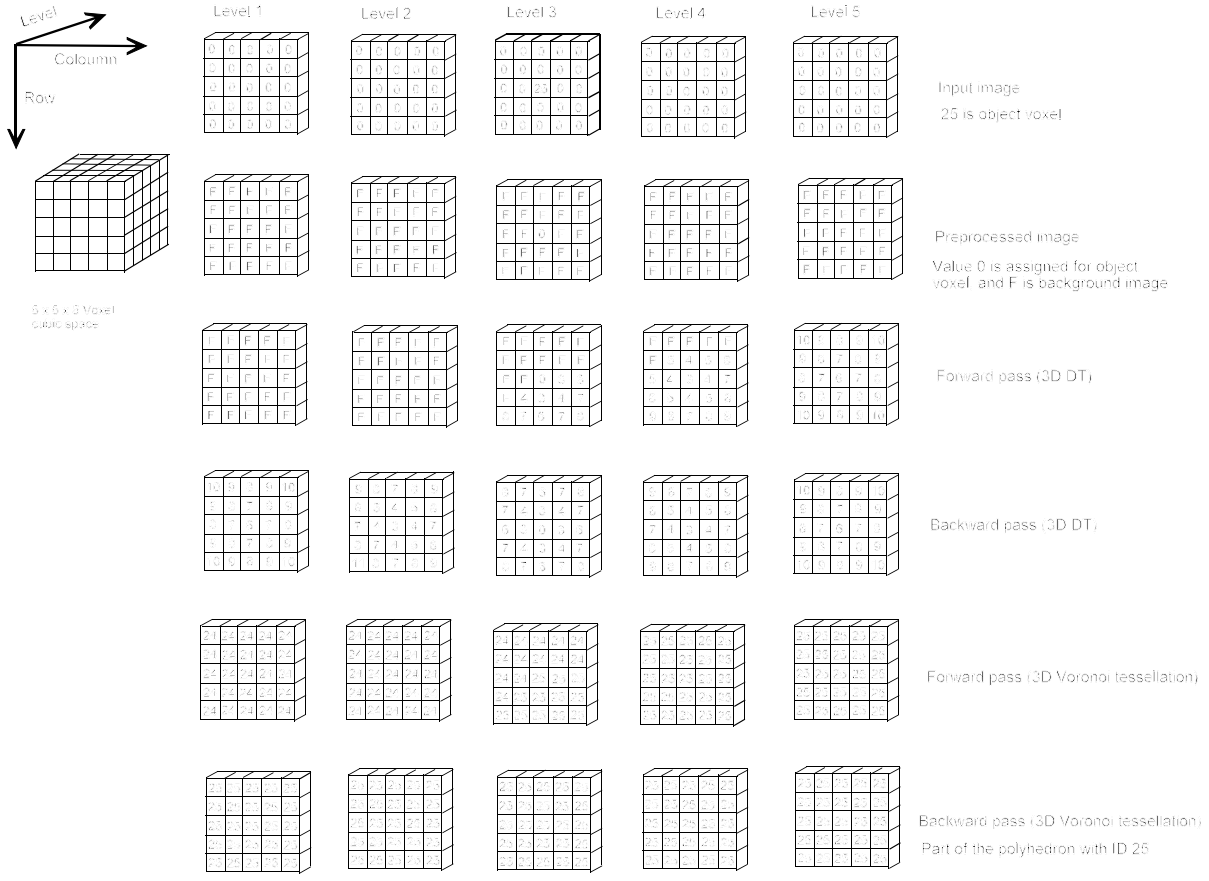


Figure 10 Slice of images (along the Z or level direction) for the 3D DT and 3D Voronoi tessellation

To generate a distance image of a 3D raster image, the first step is to set the voxel background image with highest integer value and the object voxels with zeros (i.e. 0). The image is then scanned in two passes, i.e. forward and backward passes. The forward pass (using the upper-part mask) begins from the first voxel to the last voxel. At this stage, the voxels surrounding the object voxels will get new values. The new

value is the minimum distance from the 14 possible voxel candidates. The result of the first pass is taken into account for the second pass. This time, the image is scanned with the lower-part mask (i.e. the backward pass) beginning from the last voxel and moving to the first voxel, see Figure 10 for the accumulated distance of the 5 x 5 x 5 cubic space. Finally, a 3D distance-transformed image is formed after the two passes were carried out.

3D Voronoi tessellation

A Voronoi image is generated from the DT image. Again, we generated these two images in parallel. The task also involves three steps: First, cover the image with the mask. Second, the values of the mask are added to the value of the voxels being covered by the mask. Third, a minimum value from the 14 voxel candidates is determined and assigned to the current voxel location. The original voxel value of the current voxel location is taken, assigned, and written to the 3D Voronoi file. This is done prior to the mask being moved to the next voxel location. The process continues until the last voxel of the image is reached. Again, the result of this forward pass is taken into account in the backward pass which begins from the last voxel proceeds to the first voxel of the image. Figure 10 shows how the 3D Voronoi polygons (i.e. polyhedrons) were delineated from one object voxel with ID = 25. In other words, a polyhedron of the voxels with ID 25. Visualization of the 3D DT and 3D Voronoi images or polyhedrons can be by a true 3D viewing package.

Tetrahedron Network (TEN) Generation

Using the same principle as for the 2D TIN, the algorithm for the 3D TIN utilised a mask of 2 x 2 x 2, see Figure 11. It has 8 voxel elements. It provides a unique way of establishing a tetrahedron. In order to obtain non-overlapping tetrahedrons, several predefined conditions have to be imposed during voxel scanning. There are 6 possible non-overlapping tetrahedrons that we can get from the mask. The mask is then used to scan the voxel's Voronoi tessellated-image once.

Once the tetrahedron was detected (based on the imposed conditions), it is then written to a file. The file contains a record of tetrahedrons where each record has 4 nodes, it is an ASCII file and structured as in Figure 12. Thus, it is one way of establishing a simple

tetrahedron data structure. This data structure together with a table of point coordinates provide a means for further manipulation of the data, e.g. visualization.

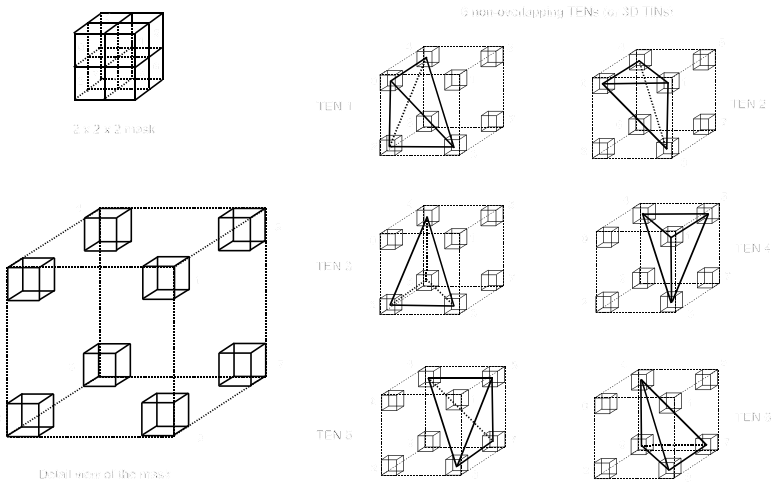


Figure 11 The six non-overlapping TENs

We implemented the algorithm and tested it by using simulated 3D raster data sets. This data set was generated by our 3D point-to-raster program. We also developed a wireframe display program for visualizing the TENs, see Figure 13 for the output display.

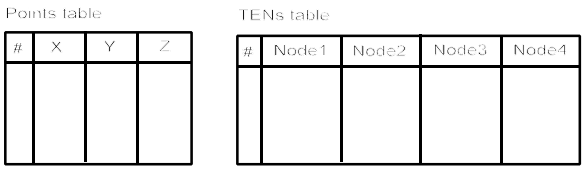


Figure 12 TEN data structure

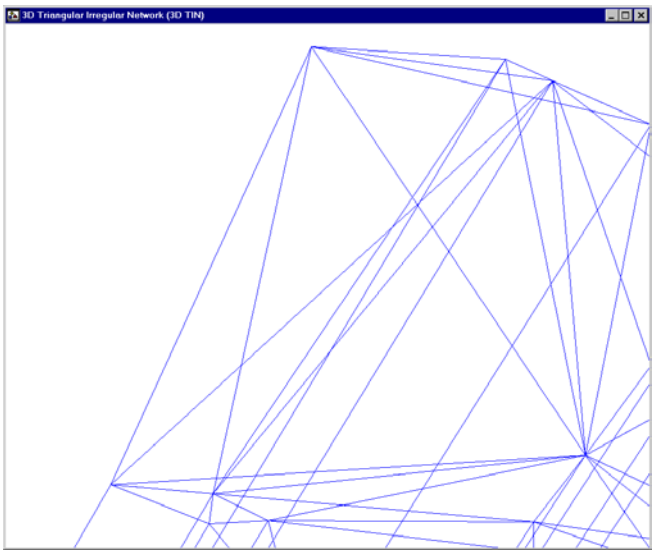


Figure 13 The TENs visualization

4.0 Object-Oriented Design of TIN-based Modelling

A data model in an information system is a collection of conceptual tools for describing data, data relationships, data semantics, data constraints and operations on data. In this section, a data model based on an object-oriented technique is described. The model realised the object-oriented concepts such as objects, encapsulation, classification, generalisation and inheritance, and aggregation. These OO terms are not described here, but the reader can consult them from literatures and various OO textbooks, e.g. Khoshafian and Abnous (1995), Isadale and Lee (1996), and Stroustrup (1997).

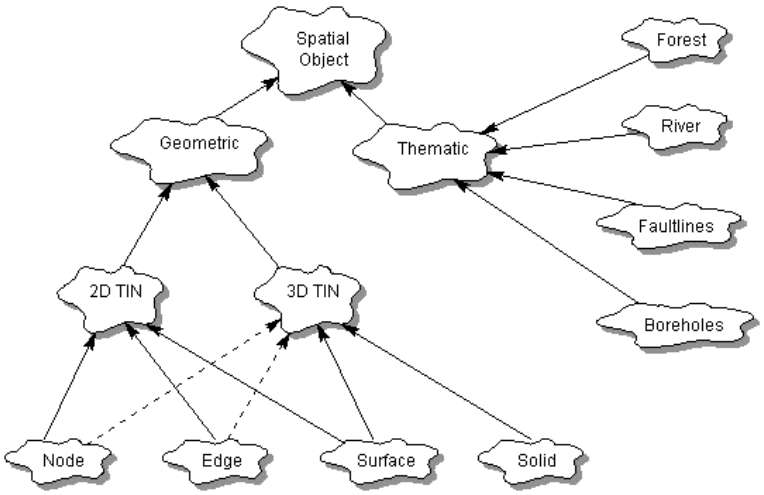


Figure 14 TINs within spatial object hierarchy

Furthermore, OO data models have been considered to satisfy many aspects of the data modelling requirements of GIS (Worboys *et al.*,1990; Webster, 1990). If we consider our previous 2D and 3D TIN modelling programs as object classes then we may relate them with other spatial domain as shown in Figure 14. Basically, a spatial object may be described by its geometric and thematic data. Then, these two data (i.e. geometric and thematic) types could be subdivided into several spatial primitives, e.g. node, edge, surface, and solid. See Figure 14 for the derivation of objects. We say a node represents point feature, an edge for line (contains at least two points), a surface for area feature (i.e. polygon), and a solid for volumetric feature. In this figure, the "Spatial object" is the base class, and two were derived classes, namely, "Geometric" and "Thematic" class. Several other classes were derived from the geometric class. Four classes were derived from the thematic class, they are "Forest", "River", "Faultlines", and "Boreholes". As for an example, the following are some definitions of the prominent classes:

class 2D TIN:
    parent class: Geometric.
    methods: e.g. 2D DT, 2D Voronoi, TIN table generation, visualization.

class Node:
    parent: 2D TIN, 3D TIN
    methods: e.g. create, delete, show identifier, GetCoordinates, etc.

class Edge:
    parent: 2D TIN, 3D TIN
    methods: e.g. create, delete, show edge, etc.

The implementation of this OO modelling including the development of the database and spatial operations become our next research task of prototyping a 3D GIS system.

The 3D GIS System

Traditionally, two-dimensional spatial data can be efficiently handled by existing GIS systems. However, the need for 3D systems is acknowledged by the GIS research community as reported in Raper (1990) and Houlding (1994). Indeed the fields of photogrammetry, simulation and modelling, mining, geology, hydrology, geo-engineering, urban and rural planning are likely to benefit from such system. The call for better geo spatial data management also provides an initiative for this so called 3D GIS. Undoubtedly, the need for such a system has been discussed in various geo-related research communities. Concepts contributing to the development of this system have been addressed in Weibel and Heller (1990), Fritsch (1996), Bruenig (1996), and Pilouk (1996). We propose the following scheme for the 3D GIS system (Figure 15):

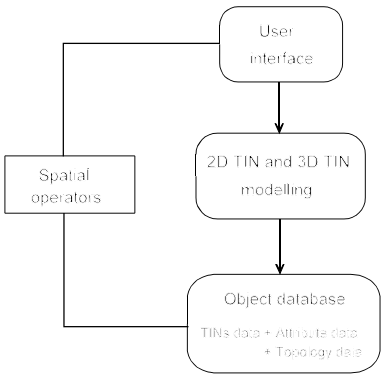


Figure 15 The possible components for 3D GIS system

1. a module supporting the user-interface
2. a module supporting 2D and 3D TIN modelling
3. a module supporting the object database, and
4. a module supporting the spatial operators.

The user interface provides a means of interaction between user and system. The 3D modelling component contains two sub-components, 2D TIN and 3D TIN. TINs have been considered as a useful data structure in 3D GIS by many researchers (Raper, 1990). Others such as Qingquan and Deren (1996) indicated that the structure was considered a powerful vector structure for 3D GIS. The structure of TEN could be organized in various ways. Figure 12 shows one of the possible structures for TEN which was developed in this research. The proposed TEN structure should be able to define the topological relationships among the primitives, i.e. point, line, surface, and solid. These TIN programs are used to generate 3D data structure from the given input of 3D datasets. The spatial operator module provides a means for spatial manipulations and analysis, e.g. 3D visualization, geometric transformation, etc. The object database is for the management of TINs data together with their attribute and topological data.

5.0 Discussion and Outlook

We have implemented and demonstrated the two programs for generating 2D and 3D triangular irregular networks. The graphics output of the TINs suggests that the algorithms work for random points in two and three-dimensional space. We also described the possible development of the TINs in object-oriented environment. The later development in OO stimulates us for further research on establishing a prototype of 3D GIS system.

References

Abdul-Rahman, A., 1992. Triangular irregular network in digital terrain relief modelling, M.Sc thesis, ITC, Enschede, The Netherlands, 80 p.

Borgefors, G., 1986. Distance transformations in digital images, Computer Vision, Graphics, and Image Processing, Vol. 34, pp. 344-371

Borgefors, G., 1996. On digital distance transformations in three dimensions, Computer Vision and Image Understanding, Vol. 64, No. 3, pp. 368-376

Bruenig, M., 1996. Integration of spatial information for geo-information systems, Lecture Notes in Earth Sciences No. 61, Springer, Berlin. 171 p.

Chen, X., K. Ikeda, K. Yamakita, and M. Nasu, 1994. Raster algorithms for generating delaunay tetrahedral tessellations, Int. Archives of Photogrammetry and Remote Sensing (ISPRS), Munich, Vol. 30, Part 3/1, pp. 124-131

Delaunay, B., 1934. Sur la sphére vide, Bulletin of the Academy of Science if the USSR, Classe. Sci. Mat., pp. 793-800

de Berg, M., 1997, Visualization of TINs, In: Algorithmic Foundations of Geographic Information Systems, Lecture Notes in Computer Science No. 1340, (eds.) van Kreveld, M., Nievergelt, J., Roos T., and Widmayer P., Springer, pp. 79-97

Fritsch, D., 1996. Three dimensional geographic information systems: status and prospects, International Archive of Photogrammetry and Remote Sensing, Vol. XXXI, Part B3, Vienna, pp. 215-221

Houlding, S. W., 1994. 3D geoscience modelling: computer techniques for geological characterization, Springer, Berlin. 309 p.

Isdale, M., and Y. C., Lee, 1996. An object-oriented modelling framework for geographic information, International Archive of Photogrammetry and Remote Sensing, Vol. XXXI, Part B4, pp. 754-758.

Khoshafian, S., and R. Abnous, 1995. Object orientation: concepts, analysis and design, languages, databases, graphical user interfaces, and standards. John Wiley, New York, 504 p.

McCullagh, M. J., and C. G. Ross, 1980. Delaunay triangulation of a random data set for isorithmic mapping, The Cartographic Journal, Vol. 17, No. 2, pp. 93-99

Mirante, A., and N. Weingarten, 1982. The radial sweep algorithm for constructing triangulated irregular networks, IEEE Transaction on Computer Graphics and Applications, Vol. 2, No. 3, pp. 11-21

Peucker, T. K., R. J. Fowler, J. J. Little, and D. M. Mark, 1978. The triangulated irregular

network, Proceedings of Digital Terrain Modelling (DTM) Symposium, American Society for Photogrammetry (ASP), St. Louis, Missouri, pp. 516-540

Pilouk, M., 1992. Fidelity improvement of DTM from contours, M.Sc. thesis, ITC, Enschede, The Netherlands, 99 p.

Pilouk, M., 1996. Integrated modelling for 3D GIS, PhD thesis, ITC Publication No. 40, 200 p.

Qingquan, L., and L. Deren, 1996. Hybrid data structure based on octree and tetrahedron in 3D GIS, Int. Archives of Photogrammetry and Remote Sensing, Vol. XXXI, Part B4, Vienna, pp. 503-507

Raper, J., 1990. Three dimensional applications in geographic information systems, Taylor and Francis, London. 189 p.

Stroustrup, B., 1997. The C++ programming language 3rd. Edition, Addison Wesley, 910 p.

Watson, D. F., 1981. Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes, The Computer Journal, Vol. 24, No. 2, pp. 167-172

Webster, C., 1990. The object oriented paradigm in GIS, International Archive of Photogrammetry and Remote Sensing, Vol. XXVIII, Part 3/2, Wuhan, pp. 947-984

Worboys, M., H. Hearnshaw, and D. J. Maguire., 1990. Object-oriented data modelling for spatial databases, International Journal of Geographical Information Systems, Vol. 4, No. 4, pp. 369-383

THE IMPLEMENTATION OF OBJECT-ORIENTED TIN-BASED SUBSYSTEMS FOR GIS

Alias Abdul-Rahman and **Jane E. Drummond**
Dept. of Geography and Topographic Science
University of Glasgow
Glasgow, G12 8QQ
Scotland, U.K

alias@geog.gla.ac.uk
jdrummond@geog.gla.ac.uk

Working Group IV

KEYWORDS: TIN, Spatial Data Modelling, Object-Oriented, and GIS

ABSTRACT

This paper focuses on the development of an object-oriented Triangular Irregular Networks (TINs) subsystems for GIS. It reviews the current development of 3D GIS which includes a discussion of the needs, demands, and related modelling and structures for a system. In the third section of the paper, we discuss the concept of object-oriented modelling and specific modelling for the TIN approach. The TIN structures are based on Delaunay triangulation which utilises the concepts of digital image transformation and Voronoi diagrams. The generated structures are used as the core data structures in the spatial modelling. The relevant aspects of TIN-based object-oriented modelling are introduced in section 4. Section five presents the implementation of the above concepts of the TIN-based spatial data in subsystems for the proposed object-oriented system in which a commercial OODBMS package is utilised. The subsystems are tested using photogrammetrically digitized datasets and presented visually by the display interface that we developed (called TINSoft). We also developed some GIS applications, e.g. contouring. Finally, we discuss some challenges and improvements needed for this kind of GIS system.

1. INTRODUCTION

The object-oriented (OO) approach is now being utilised in various fields including GIS. In GIS, OO techniques are used for various tasks including spatial data modelling, databasing, and system development. A research trend involving a shift from structural to OO techniques is very much in evidence in the GIS community. This trend is mainly due to the strength of the OO approach over the traditional structural technique of programming and system development (Egenhofer and Frank, 1989; and Worboys, 1995), that is OO modelling only requires representation much closer to the real world than are needed in the more abstract structured approach. Complex spatial data handling as required in GIS also contributes to this paradigm shift (Webster, 1990) as traditional relational databases could not handle efficiently such complex spatial data. It is the subject of this paper to further investigate the use of an OO system where TIN spatial data is the data structure of concern. Other work carried out by the first author is also investigating the use of TEN (tetrahedral networks) data structures.

This paper discusses some 3D GIS issues, the needs of the system, and the related problems of modelling and data structuring in section 2. OO conceptual modelling is discussed in section 3. We discuss TIN spatial data modelling in section 4, and the TIN-based subsystems for an information system in section 5. Finally, we present suggestions to improve the proposed subsystems in the summary.

2. 3D GIS

In this section we review and discuss some problems and related issues in 3D GIS software development. In GIS, 2D systems are common, widely used, and able to handle most of the GIS tasks efficiently. The same kind of system may not be able to handle 3D data as more advanced 3D applications are demanded (Raper and Kelk, 1991; Li, 1994) such as representing the full length and width of a borehole. Some examples of 3D applications areas are listed in section 2.1. 3D GIS is very much needed to generate information from such 3D data. The system is not a simple extension of another dimension (i.e. the 3rd dimension) on to 2D GIS. To add this third dimension into existing 2D GIS needs a thorough investigation of many aspects of GIS including a different concept of modelling, representation,

and aspects of data structuring. Existing GIS packages are widely used and understood for handling, storing, manipulating, and analysing 2D spatial data. Their capability and performance for 2D and for 2.5D data (DTM) is generally accepted by the GIS community. Should we consider a GIS package which can handle and manipulate 2D data and DTM data as a 3D GIS system? The answer is no because DTM data is not real 3D spatial data. The third dimension of the DTM data only provides a surface attribute to the planimetric data of x, y coordinates. In fact, we hardly find any current GIS software to be in a position to handle real 3D spatial data. Although the problem has been addressed by several researchers such as Raper and Kelk (1991), Cambray (1993), Li (1994), Pilouk (1996), and Fritsch (1996), some further aspects particularly spatial data modelling using OO techniques need to be investigated. This modelling issue is addressed in this paper, see section 3 and section 4. The demand for this kind of system is discussed in the next section.

2.1 Who needs 3D GIS

As in the popular 2D GIS for 2D spatial data, 3D GIS is for managing 3D spatial data. Raper and Kelk (1991), Li (1994), Förstner (1995), and Bonham-Carter (1996) described some of the three dimensional applications in GIS, including:

- ecological studies
- environmental monitoring
- geological analysis
- civil engineering
- mining exploration
- architecture
- automatic vehicle navigation
- archeology
- 3D urban mapping
- landscape planning

The above applications may produce much better information if they were handled in a 3D spatial system. It appears that complex 3D spatial objects on the surface and subsurface demand better solutions (e.g. in terms of modelling, analysis, and visualization) than the existing systems can offer.

The next section reviews the modelling and data structures contributing to 3D GIS.

2.2 Modelling and Data Structuring

Much previous work done on 3D data modelling concentrated on the use of voxel data structures (Jones, 1989). This particular approach does not address spatial modelling aspects, it is only useful for the reconstruction of 3D solid objects and for some basic geometric computations. One of the problems with this model is that it needs very large computer space and memory.

Carlson (1987) proposed a model called simplicial complex. He used the term 0-simplex, 1-simplex, 2-simplex, and 3-simplex to denominate spatial objects of node, line, surface, and volume. His model can be extended to *n*-dimensions.

While Cambray (1993) proposed CAD models for 3D objects combined with DTM as away to create 3D GIS, that is a combination of Constructive Solid Geometry (CSG) and Boundary representation (B-rep).

Other attempts to develop 3D GIS can be found in Kraus (1995), Fritsch and Schmidt (1995), and Pilouk (1996). These attempts were based on the TIN data structure to represent 3D terrain objects but no reports exist on the any related aspects of OO technique for the modelling and data structure.

Data modelling and structuring of 3D spatial objects in GIS has not been as successfully achieved as in CAD (Li, 1994). Data modelling in GIS is not only concerned with the geometric and attribute aspects of the data, but also the topological relationships of the data. Topology of spatial data must be available so that the neighbouring objects can be determined. There are a number of mathematical possibilities for the determination of the topological description of objects. Within the TIN data structure, we have developed a program to determine the neighbouring triangles. The information gained from the generated TIN's neighbours is useful for further spatial analysis and applications. We also established topological relationships for linear objects as represented by TIN edges. One edge is represented by a start node and an end node. From this edge topology, a chain of edges or arcs could be easily established. For TIN data, another approach is the simplicial complex developed by Carlson. A TIN's node is equivalent to 0-simplex, TIN's edge is equivalent to 1-simplex, a TIN surface (area) is equal to 2-simplex, and 3-simplex is equivalent to a 3D TIN (tetrahedron). The simplicial complex technique checks the consistency of generated TIN structures by Euler's equality formulae, see Carlson (1987) for a detailed discussion. We explain our OO TIN approach in section 4 after elaborating OO conceptual modelling.

3. OBJECT-ORIENTED CONCEPTUAL MODELLING

Object-oriented conceptual modelling is now widely utilised in many fields including GIS. The concepts of OO such as object classification, encapsulation, inheritance, and polymorphism are able to ease the modelling of complex real world objects.

3.1 Object-Oriented Concepts

As mentioned above the object-oriented approach is now being promoted as the most appropriate method for modelling complex situations that are concerned with real-world phenomena, and thus applicable to GIS. Object-oriented concepts are considered more flexible and powerful than the traditional structural programming and other major database models such as the relational or entity-relationship model. Object-oriented concepts contribute to modelling as follows:

- Objects and abstraction mechanisms (classification, generation, aggregation, and association). These aspects of OO can be used for modelling real world phenomena, e.g. modelling of spatial data for geoinformation systems.
- Inheritance, propagation, encapsulation, persistence, Abstract Data Type (ADT), polymorphism, and overloading. These aspects of OO can be used to construct and implement the model discussed in (a).

The usefulness of these concepts in spatial modelling are explained below.

3.2 The Abstraction Mechanisms

Data abstraction is a method of modelling data. Object-oriented design uses four major abstraction mechanisms: (1) classification, (2) generalization, (3) inheritance, and (4) polymorphism. In object-oriented programming, any physical or logical entity in the model is an "object". The definition of a type of object is called a "class", and each particular object of that type known as an "instance" of the class. Once a class has been defined, it can, potentially be reused in other programs by simply including the class definition in the new program. However, it is not necessary for the programmer who uses a class to know how it works, they simply need to know how to use it. The definition of operations on or between objects are called "methods", and the invocation of methods is referred to as "passing a message". Recent research in software engineering has promoted an object-oriented design method by which real world objects and their relevant operations are modelled in a program which is more flexible and better suited to describe complex real world situations (Khoshafian and Abnous, 1995). We may also consider object-orientation as a particular view of the world which attempts to model reality as closely as possible (Webster, 1990). Details on all relevant OO concepts (object, abstraction, data types, class hierarchy, inheritance, classification, aggregation, generalization and association) can be found in the OO literature such as Booch (1990), Bhalla (1991), and Stroustrup (1997). The following are some OO terms:

Classification
Classification can be expressed as the mapping of several objects (instances) onto a common class. In object-oriented approach, every object is an instance of a class (a class is a fundamental building block in OO language). Class describes the common features of a set of objects with the same characteristics; it also defines the nature of the state and behaviour, while an object records the identity and state of one particular instance of a class. Abstract Data Type (ADT) is a mechanism to create a class of spatial objects or any class in a domain of objects. An object is a basic run-time entity in an object-oriented system. This entity includes data and procedures that operate on data. Viewed from a programming stand point, objects are the elements of an OO programming system sending and receiving messages.

Generalization
Generalization in OO provides for the grouping of classes of objects, which have some operations in common, into a more general superclass. Objects of superclass and subclass are related by an *"is a"*- relation, since the object of a subclass is also an instance of a superclass.

Inheritance
Inheritance allows the building a hierarchy of types or classes that best describes the real world situation in the application field. Each class can take all or part of the structural or behavioural features from other classes, which are its parents. In turn, the newly defined class is a child of the classes from which it has inherited its features.

Inheritance helps in deriving application-oriented classes without starting every definition from scratch. Also, it makes it easier to create logically complex classes from simpler classes.

Polymorphism
Polymorphism is a mechanism to define the different actions of the same named function on different classes. It is implemented by inheriting some functions from parent classes and overriding or modifying part of them. Usually, the newly created class has similar but not the same behaviour as its parents for that functional aspect. Polymorphism provides great flexibility in class derivation, for example, perimeter operation may have different implementations for different classes such class "area", class "triangle", class "polygon", etc. Each class performs the perimeter operation differently although it has the same function name.

4. OBJECT-ORIENTED TIN SPATIAL DATA MODELLING

In this section we provide a discussion of the OO TINs spatial data modelling techniques. Conceptually, the general modelling steps as depicted in Figure 1 could be used for TIN spatial data modelling. That is, the three-step approach, namely the conceptual, the logical, and the physical steps. The class schema for spatial data modelling are described below.

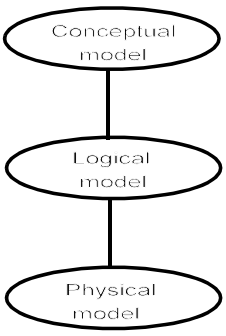
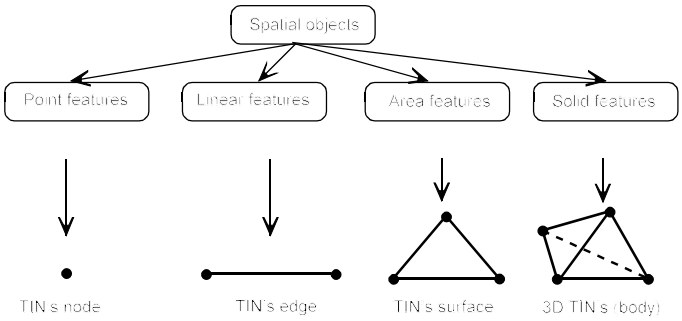


Figure 1 A typical spatial data modelling steps

4.1 The Class Schema
The schema is based on several classes, they are Spatial Objects (the super class), and four major subclasses which are Node, Edge, Polygon, and Solid.

Spatial objects
The spatial object class is a general class of the real world objects. It is the super class in the class hierarchy. We assume that all other objects are derived from this super class, see Figure 3. All terrain objects could be categorised into several sub classes such as points, lines, areas, and solids (volume) features. In OO modelling, these feature types are the classes in the modelling hierarchy.

Node
A node can be considered as the most basic geometrical unit in spatial data modelling. It may represent point entities or point objects at a particular mapping scale. Examples of point objects are wells, terrain spot heights, and the like. In geoinformation, we may represent these objects by a class called a node class. The coordinates of the nodes (including the nodes represent edges) are held by a coordinates container class, called XYZContainer class.



Edge
An edge can be represented by two nodes at each end (i.e., a start node and end node). In this study we consider two end points make a straight edge. We used this edge type to represent linear features. The arc container class, called ARCContainer holds all the arcs. The arcs container also serve any other class which requires arcs data in their operations for example the polygon class needs the arcs in order to form polygons.

Figure 2 TINs representations for spatial objects

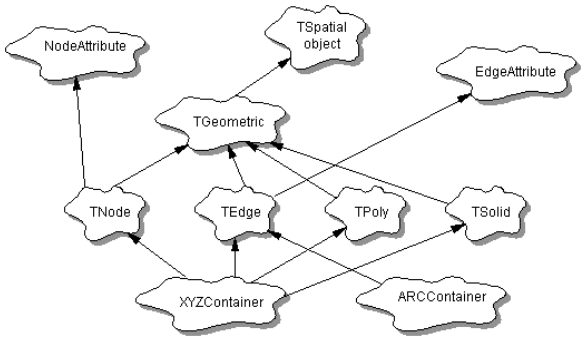


Figure 3 The class diagram (using the Booch notation)

Polygon
A polygon (sometimes known as a surface) is used to represent area features such as lakes, ponds, etc. A polygon may be constructed by chains of closed edges.

Solid (or Body)
This is a representation for solid or body features such as buildings, trees. A chain of points and lines form body objects for example, 3D TIN can be represented by a series of triangle nodes and edges as indicated in Figure 2.

The class schema in Figure 3, depicted using Booch (1990) notation is the representation of the TIN spatial data model. The schema has: four geometric classes namely (TNode, TEdge, TPoly, and Tsolid); two types of containers (geometry and attribute). The geometric containers contain the XYZ locations whereas the attribute containers are for the thematic values, e.g. names.

4.2 The POET OO Database Development

The DBMS is used to generate the OO database from the constructed TIN spatial data. In this work the schema needs to be modelled according to the POET database model (POET, 1996), that is it is required to construct all the C++ classes as classes which POET can understand. In this case, all the classes in the schema have to be compiled by the POET PTXX compiler. The PTXX compiler maps all the normal C++ classes into the several relevant PTXX schema files which in turn are used for writing application programs (runs under normal C++ compiler) as well as for populating the database. The PTXX complier also generates the OO database from the schema, see Figure 4. For database query, the OQL (Object Query Language) syntax is used, see POET (1996) for the details of the language. An example of a query which can be performed from the database is:

```
defined extent allTEdge for TEdge;
select Edge
from Edge in allTEdge
where Edge.EdgeAtr.EdgeName = "River*"
```

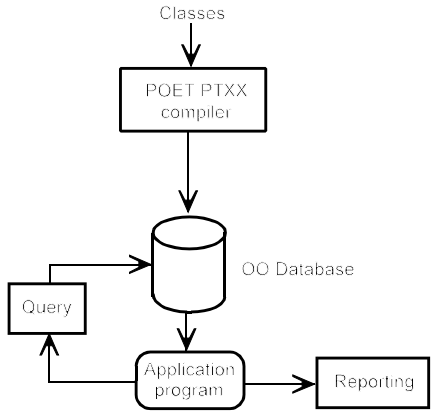


Figure 4 The POET database development flows

5. OBJECT-ORIENTED TIN-BASED SUBSYSTEMS FOR GIS

5.1 The Subsystems

The OO TIN GIS is based on several fundamental concepts and aspects of spatial data which have been discussed in the previous sections. Basic components in the system are data input processing, TIN data construction, TINs database, transformation operations, data output, and user-interface. Rasterization forms a major operation in the data input component. Figure 5 shows the other major components of the proposed system which includes the use of other commercial software, i.e. ILWIS™ (Integrated

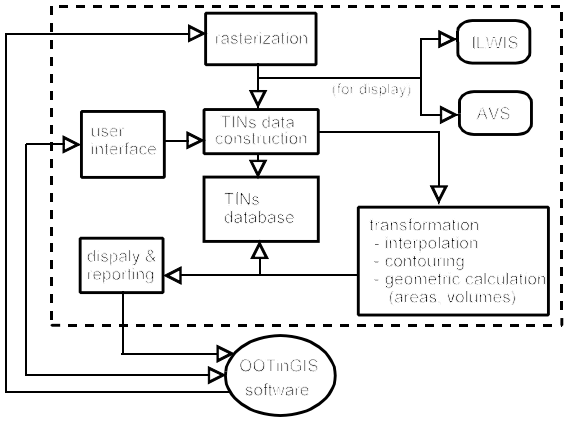


Figure 5 The proposed system for the TIN-based spatial data

Land and Water Information System) and AVS™ (Advanced Visualization System). These two packages are only for display purposes especially for validating the output from the rasterization process. We also developed a simple user interface as part of the software development. Besides our own written programs for databasing purposes, we also used a commercial database package, called POET™ OODBMS as mentioned. The DBMS package is for the development of the OO TIN spatial database.

5.2 The Test and Results

We tested the subsystems (or their components) by using photogrammetrically digitized datasets (other types of datasets such as data acquired by field survey, or by map digitizing are also possible as input to the subsystems). Figure 6 shows the study area (Drumbuie, near Kyle of Lochash, north-west Scotland). Other diagrams visually illustrate some of the output from the subsystems.

Figure 6 The study area (orthophoto image) from where points and lines were extracted using 3D digitizing (with stereo mate)

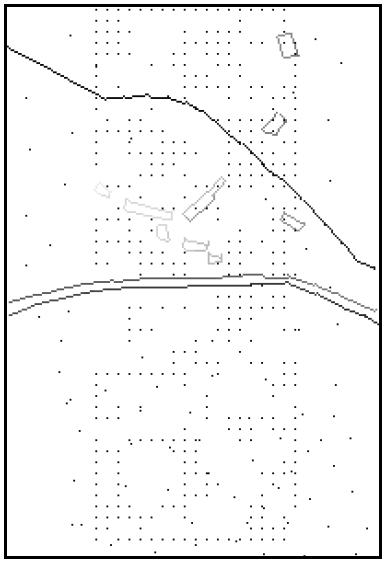


Figure 7 The rasterised points and lines of selected features

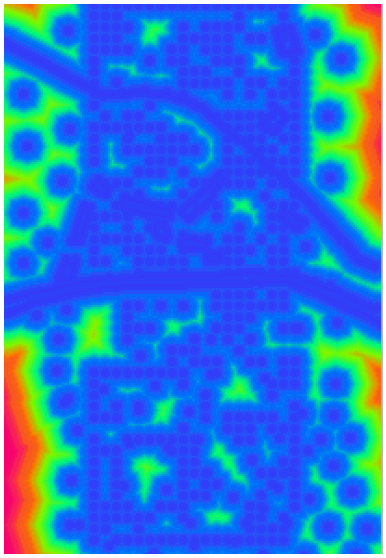


Figure 8 The distance transform (DT) image of the area

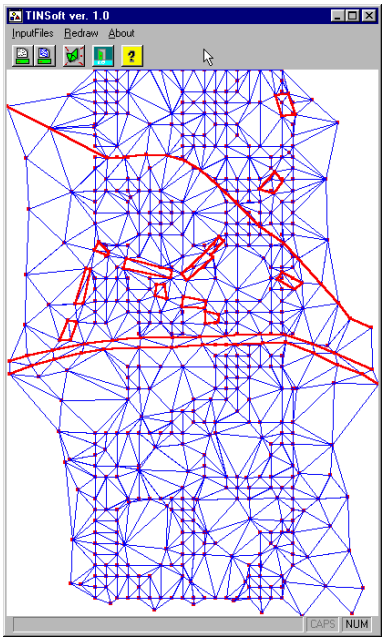


Figure 9 The generated TINs incorporated with constrained edges.

Figure 7 shows the output from the rasterization component. Figure 8 is the intermediate process of 2D TIN data construction component, that is the distance transformation computation output. The generated TINs are in Figure 9 whereas Figure 10, the simple display interface, illustrates the perspective view of the 3D objects such as buildings and trees sitting on top of the TIN surfaces. Contours of the terrain surfaces could also be derived from the TIN database as clearly draped on the terrain surface.

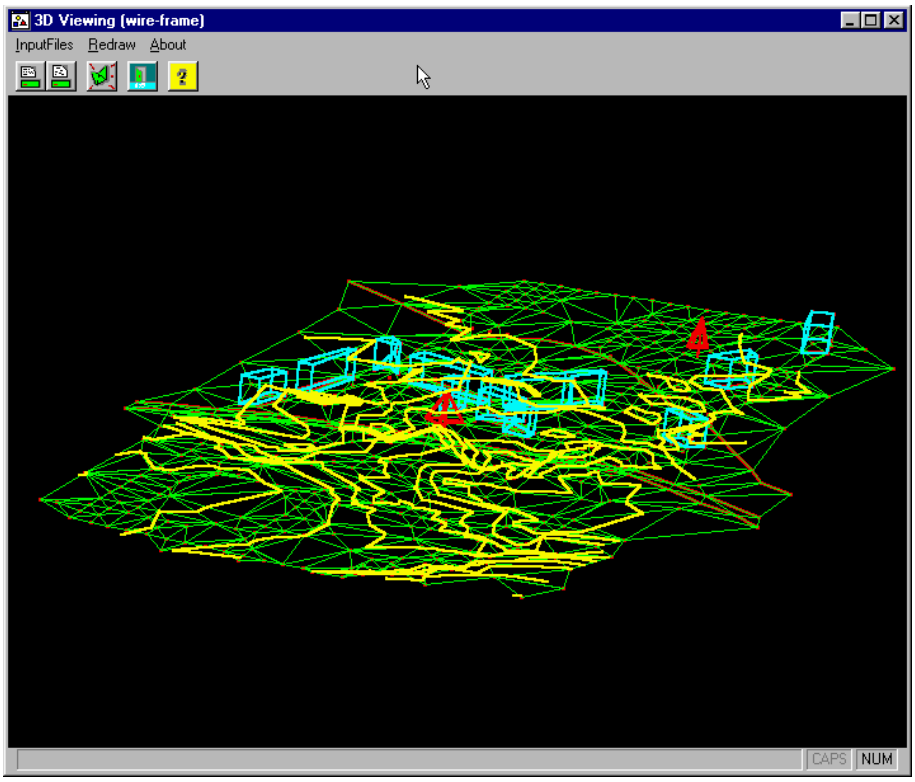


Figure 10 Perspective view of 3D objects (buildings) and trees draped on TINs surfaces with the derived contours (4-metre intervals).

6. SUMMARY

We have introduced several subsystems for OO TIN-based GIS. The generated products (i.e. the output) from the subsystems indicate the workability of these components. Although the system is not yet complete as one whole GIS system, but the results produced from the components are promising. An integration of the subsystems with the commercial OODBMS needs to be investigated so that seamless integration could be achieved. Thus, from the perspective of GIS software development, the object-oriented TIN data modelling can be used to develop a geo information system.

ACKNOWLEDGEMENTS

Thanks are due to the following organisations which have contributed one way or another in our research work, they are LH System Inc. (Helava-Leica) for the digital photogrammetric software, Advanced Visualization Systems Inc., (AVS) for 3D raster data visualization package, and POET Inc. The generous assistance from Dr. Ron Poet of Glasgow University's Department of Computer Science and Mr. Prasad Jeevanigi on POET's compiler error messages must also be acknowledged.

REFERENCES

Bhala, N., 1991. Object-oriented data models: a perspective and comparative review. Journal of Information Science, Vol. 17, pp. 145-160.

Bonham-Carter, G. F., 1996. Geographic information systems for geoscientists: modelling with GIS. Computer Methods in the Geosciences, Vol. 13, Pergamon Publications, 398 p.

Booch, G., 1990. Object oriented analysis and design with applications. 2nd. Edition, Addison-Wesley, 589 p.

Bric, V., 1993. 3D vector data structures and modelling of simple objects in GIS. M. Sc. Thesis, ITC, Enschede, The Netherlands, 107 p.

Cambray, de B., 1993. Three-dimensional (3D) modelling in a geographical database. Proceedings of AutoCarto 11, Minneapolis, USA, pp. 338-347.

Egenhofer, M. and Frank, A. U., 1989. Object-oriented modelling in GIS: inheritance and propagation. Proceedings of AutoCarto 9, Baltimore, pp. 588-598.

Förstner, W., 1995. GIS - the third dimension. Workshop on Current Status and Challenges of Geoinformation Systems, IUSM working group on LIS/GIS, University of Hannover, Germany, pp. 65-72.

Fritsch, D., 1996. Three-dimensional geographic information systems - status and prospects. International Archives of Photogrammetry and Remote Sensing (ISPRS), Vienna, Austria, Vol. 31, Part 4, pp. 215-221.

Fritsch, D. and Schmidt, D., 1995. The object-oriented DTM in GIS. Proceeding of 45th. Photogrammetric Week, Stuttgart, pp. 29-34.

Jones, C. B., 1989. Data structures for three-dimensional spatial information systems in geology. International Journal of Geographic Information System (IJGIS), Vol. 1, no. 3, pp. 15-31

Khoshafian, S., and Abnous, R., 1995. Object orientation : concepts , analysis, languages, databases, graphical user interfaces, standards, Second Edition, John Wiley, 504 p.

Kraus, K., 1995. From digital elevation model to topographic information system. Proceeding of 45th.Photogrammetric Week, D. Fritsch and D. Hubbie (eds.), Stuttgart, Germany, pp. 277-285.

Li, R., 1994. Data structures and application issues in 3D geographic information systems. Geomatica, Vol. 48, No. 3, pp. 209-224.

Pilouk, M., 1996. Integrated modelling for 3D GIS. International Institute of Aerospace Survey and Earth Sciences (ITC), Publication No. 40, Enschede, The Netherlands, 200 p.

POET , 1996. POET™ C++ programmer's guide. Release 4, 560 p.
http://www.poet.com

Raper, J. and Kelk, B., 1991. Three-dimensional GIS. In: Geographical Information Systems: Principles and Applications. D. J., Maguire, M. Goodchild and D. Rhind (edts.), Longman Geoinformation, pp. 219-317.

Stroustrup, B., 1997. The C++ programming language, 3rd. Edition. Addison-Wesley, 910 p.

Webster, C., 1990. The object-oriented paradigm in GIS. International Archive of Photogrammetry and Remote Sensing (IAPRS), Vol. 28, Part 3/2, Comm. III, Wuhan, China.

Worboys, M. F., 1995. GIS: a computing perspective. Taylor and Francis Publication, 376 p.

Volume II

Software Code

This is the Volume II of this thesis. It lists all the C++ codes for all the software developed in this research using Borland C++ version 5.02 compiler.

2D Rasterization Code

The following gives the complete code for the 2D rasterization program (called 2D RASTER).

```
 //*********************************************************************////
//      A Program to rasterize points in 2D                                 //
//                Input: XY world coordinates                               //
//                Output: MPD and MPI ILWIS format                          //
//                                                                          //
    Copyright (c) Alias Abdul-Rahman, 1999
// *********************************************************************//

#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <stdio.h>
#include <io.h>
#include <conio.h>
#include <iomanip.h>
#include <string>

const int rowsize = 800;
const int colsize = 800;

typedef struct PointStruct                    // structure for XY world
coordinates
  {
   double x;
   double y;
   double z;
  } PointType;

PointType Point;

typedef short Image[rowsize][colsize];    // Image: Pixel variable

// the MPI file structure
typedef struct MpiStruct
          {
          short Nscanlines;      // no. of image rows
          short Npixels;         // no. of image columns
          short Pvmin;           // pixel's minimum value
          short Pvmax;           // pixel's maximum value
          short Maptype;         // map type
          short Patch;           // patch type
          short Scale;           // scale factor
          short Crdtype;         // coordinate type
          float a11;             // transformation coefficient
          float a12;             //       "
          float a21;             //       "
          float a22;             //       "
          float b1;              //       "
```

```
          float b2;              //       "
          } MpiType;

MpiType MPI;

Image Pixel;

int xlength, ylength;
int pixsize;
short maxrow, maxcol;
short npnt, pnt, maxpnt;
float scalex, scaley, scale;
double xmin, ymin, xmax, ymax;
FILE* MPDfile;
FILE* MPIfile;


// prototype section
void GetXYZfile(double&, double&, double&, double&);
void WriteMPIfile();
void GetPixelSize(int&);
void GetNumRowCol(int, int&, int&);
void MakeRaster(PointType, int, int, int);
void DisplayPixel();
void PressAnyKey();

// Main
void main()
  {
   int i;

   GetXYZfile(xmin, ymin, xmax, ymax);
   GetPixelSize(pixsize);
   GetNumRowCol(pixsize, xlength, ylength);
   MakeRaster(Point, pixsize, xlength, ylength);
   WriteMPIfile();
   DisplayPixel();
   PressAnyKey();

  }

// definition section
void GetXYZfile(double& xmin, double& ymin, double& xmax, double& ymax)
  {
   string charX, charY, charZ;

   ifstream XYZfile("c:\\data\\lochgrd.xyz");

     if (! XYZfile)
       {
        cerr << "Point file not found" << endl;
        exit(1);
       }

   // reads the file header
   XYZfile >> charX >> charY >> charZ;

   npnt = 1;
   XYZfile >> Point.x >> Point.y >> Point.z;
```

```
   // to calculate min xy, max xy
   xmin = Point.x;
   ymin = Point.y;
   xmax = Point.x;
   ymax = Point.y;

   do
     {
      printf("\r");
      cout << "Reading points ... " << npnt;

      XYZfile >> Point.x >> Point.y >> Point.z;
        if (xmin > Point.x)
            xmin = Point.x;

        if (ymin > Point.y)
            ymin = Point.y;

        if (xmax < Point.x)
            xmax = Point.x;

        if (ymax < Point.y)
            ymax = Point.y;

     npnt ++;
     } while (! XYZfile.eof());

   xlength = xmax - xmin;
   ylength = ymax - ymin;
   maxpnt = npnt - 1;
   XYZfile.close();

  }

void GetPixelSize(int& pixsize)
  {
   cout << endl;
   cout << "Enter pixel size (in meter): " << endl;
   cin >> pixsize;
  }

void GetNumRowCol(int pixsize, int& xlength, int& ylength)
  {
   xlength = xmax - xmin;
   ylength = ymax - ymin;
   maxrow = int (ylength / pixsize) + 1;
   maxcol = ceil(int (xlength / pixsize)) + 1;
  }

void MakeRaster(PointType Point, int pixsize, int xlength, int ylength)
  {
   int i, r, c, j;
   short PointID;
   short row, col;
   long cursor;
   float scale;
   string charX, charY, charZ;

   scale = 1.0 / float(pixsize);

   cout << endl;
```

```
cout << "xmin, ymin: " << xmin << ", " << ymin << endl;
cout << "xmax, ymax: " << xmax << ", " << ymax << endl;
cout << "pix size : " << pixsize << endl;
cout << "xlength : " << xlength << endl;
cout << "ylength : " << ylength << endl;
cout << "xmin :" << xmin << endl;
cout << "ymin :" << ymin << endl;
cout << "maxrow :" << maxrow << endl;
cout << "maxcol :" << maxcol << endl;

cout << setiosflags(ios::showpoint) << endl;
cout << setprecision(8) << "scale   : " << scale << endl;

ifstream XYZfile("c:\\data\\lochgrd.xyz");

// reads the file header
XYZfile >> charX >> charY >> charZ;

if (! XYZfile)
    {
     cerr << "Point file not found" << endl;
     exit(1);
    }

FILE* MPDfile;
char mpdfilename[] = "c:\\data\\lochgrd.MPD";
MPDfile = fopen(mpdfilename, "wb");

pnt = 1;

while (! XYZfile.eof())
  {
   XYZfile >> Point.x >> Point.y >> Point.z;
   col = int(scale * (Point.x - xmin)) + 1;
   row = int((-1 * scale) * (Point.y - ymin)) + maxrow + 1;
   if (XYZfile.eof() == true)
     pnt = pnt - 1;
   PointID = pnt;
   Pixel[row][col] = PointID;
   printf("\r");
   cout << "Rasterizing the points ... " << pnt;
   pnt ++;
  }

pnt = pnt-1;
maxpnt = pnt;

for (row = 1; row <= maxrow+1; row ++)
  {
   fwrite(Pixel[row], sizeof(short)* maxcol, 1, MPDfile);
  }

cout << endl;
cout << endl;
cout << " Total points : " << maxpnt;

XYZfile.close();
fclose(MPDfile);

}
```

```
void WriteMPIfile()
  {
   FILE* MPIfile;
   char mpifilename[] = "c:\\data\\lochgrd.MPI";
   MPIfile = fopen(mpifilename, "wb");

   // assign the values for the MPI file
   MPI.Nscanlines = maxrow + 1;
   MPI.Npixels = maxcol;
   MPI.Pvmin = 1;
   MPI.Pvmax = maxpnt;
   MPI.Maptype = 2;
   MPI.Patch = 0;
   MPI.Scale = 0;
   MPI.Crdtype = 1;
   MPI.a11 = scale;
   MPI.a12 = 0;
   MPI.a21 = 0;
   MPI.a22 = -scale;
   MPI.b1 = 1;
   MPI.b2 = maxrow + 1;

   // write out the above values in the MPI file
   fwrite(&MPI, sizeof(MpiType), 1, MPIfile);
   fclose(MPIfile);
  }


void DisplayPixel()
  {
   int i, j;
   cout << endl;
   cout << endl;
   cout << "Displaying pixel value ..." << endl;
   cout << endl;

   for (i = 0; i <= maxrow; i ++)
     {
      //for (j = 0; j <= maxcol; j ++)
        //{
         printf("\r");
         cout << "[row]: " << i;

         //cout << "Pixel[row][col]: Pixel[" << i << "][" << j <<
                //"]: " << Pixel[i][j];

        //}
     }
  }
```

```
void PressAnyKey()
  {
   cout << endl;
   cout << endl;
   cout << "Press any key to continue" << endl;
   getch();
   cout << endl;
  }
```

Constrained 2D Rasterization Code

The following gives the complete code for the constrained 2D rasterization program.

```
//*************************************************************************///
//      A Program to rasterize points and edges or arcs in 2D              //
//                Input: XYZ coordinates                                   //
//               Output: MPD and MPI ILWIS format                          //
//                                                                         //
    Copyright (c) Alias Abdul-Rahman, 1999
// //*************************************************************************/

#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <stdio.h>
#include <io.h>
#include <conio.h>
#include <iomanip.h>
#include <string>

const int rowsize = 800;
const int colsize = 800;
const int maxarc = 200;
const int maxpoint = 1000;

typedef struct PointStruct         // structure for XY world coordinates
  {
   float x;
   float y;
   float z;
  } PointType;
PointType* Point[maxpoint];

typedef struct ArcStruct           // structure for Arc table
  {
   int StartNode;
   int EndNode;
  } ArcType;
ArcType* Arc[maxarc];

typedef short Image[rowsize][colsize];    // Image: Pixel variable

// the MPI file structure
typedef struct MpiStruct
          {
          short Nscanlines;      // no. of image rows
          short Npixels;         // no. of image columns
          short Pvmin;           // pixel's minimum value
          short Pvmax;           // pixel's maximum value
          short Maptype;         // map type
```

```
          short Patch;           // patch type
          short Scale;           // scale factor
          short Crdtype;         // coordinate type
          float a11;             // transformation coefficient
          float a12;             //       "
          float a21;             //       "
          float a22;             //       "
          float b1;              //       "
          float b2;              //       "
          } MpiType;

MpiType MPI;
Image Pixel;

int numarc;
float xlength, ylength;
int pixsize;
short maxrow, maxcol;
short npnt, pnt, maxpnt;
float scalex, scaley;
float scale;
float xmin, ymin, xmax, ymax;
FILE* MPDfile;
FILE* MPIfile;


// prototype section
void GetXYZfile();
void GetXYMinMax(float&, float&, float&, float&);
void GetArcFile();
void GetMiddleXY(float, float, float, float, float&, float&);
void GetXYZforArcNodes(int, float&, float&, float&, float&,
                       short&, short&);
void WriteMPIfile();
void GetPixelSize(int&);
void GetNumRowCol(int, float&, float&);
void MakeRaster(int, float, float);
void WriteToFile(Image Pixel);
void DisplayPixel();
void PressAnyKey();
void AllocateMemory();
void DeallocateMemory();

// Main
void main()
  {
   int i;
   AllocateMemory();
   GetXYZfile();
   GetXYMinMax(xmin, ymin, xmax, ymax);
   GetPixelSize(pixsize);
   GetNumRowCol(pixsize, xlength, ylength);
   MakeRaster(pixsize, xlength, ylength);
   WriteMPIfile();
   DisplayPixel();
   DeallocateMemory();
   PressAnyKey();
  }

// Implementation section
```

```
void AllocateMemory()
  {
   for (int i = 0; i < maxpoint; i ++)
     Point[i] = new PointType;

   for (int ii = 0; ii < maxarc; ii ++)
     Arc[ii] = new ArcType;
  }

void DeallocateMemory()
  {
   for (int i = 0; i < maxpoint; i ++)
     delete [] Point[i];

   for (int ii = 0; ii < maxarc; ii ++)
     delete [] Arc[ii];
  }

void GetXYZfile()
  {
   string charX, charY, charZ;

   //ifstream XYZfile("c:\\data\\lochass.xyz");
   //ifstream XYZfile("c:\\data\\small.xyz");
   ifstream XYZfile("c:\\data\\p2.xyz");

     if (! XYZfile)
       {
        cerr << "Point file not found" << endl;
        exit(1);
       }

   // reads the file header
   XYZfile >> charX >> charY >> charZ;

   int npnt = 1;
   while (! XYZfile.eof())
     {
      XYZfile >> Point[npnt]->x >> Point[npnt]->y >> Point[npnt]->z;
      printf("\r");
      cout << "Reading points ... " << (npnt-1);
      npnt ++;
     }

  }
void GetXYMinMax(float& xmin, float& ymin, float& xmax, float& ymax)
  {
   string charX, charY, charZ;
   float x, y, z;

   //ifstream XYZfile("c:\\data\\lochass.xyz");
   //ifstream XYZfile("c:\\data\\small.xyz");
   ifstream XYZfile("c:\\data\\p2.xyz");

     if (! XYZfile)
       {
        cerr << "Point file not found" << endl;
        exit(1);
       }
```

```
   // reads the file header
   XYZfile >> charX >> charY >> charZ;

   npnt = 1;
   XYZfile >> x >> y >> z;

   // to calculate min xy, max xy
   xmin = x;
   ymin = y;
   xmax = x;
   ymax = y;

   while (! XYZfile.eof())
     {
      //npnt ++;
      XYZfile >> x >> y >> z;

        if (xmin > x)
            xmin = x;

        if (ymin > y)
            ymin = y;

        if (xmax < x)
            xmax = x;

        if (ymax < y)
            ymax = y;

      npnt ++;
     }

   xlength = xmax - xmin;
   ylength = ymax - ymin;
   maxpnt = npnt - 1;
   XYZfile.close();

  }

void GetArcFile(int& numarc)
  {
   string charsnode, charenode;
   //ifstream ARCfile("c:\\data\\lake.arc");
   //ifstream ARCfile("c:\\data\\small.arc");
   ifstream ARCfile("c:\\data\\p2.arc");

     if (! ARCfile)
       {
        cerr << "Arc file not found" << endl;
        exit(1);
       }

   // reads the file header
   ARCfile >> charsnode >> charenode;

   int narc = 0;
   while (! ARCfile.eof())
     {
      narc ++;
      printf("\r");
      cout << "Reading arcs ... " << narc;
```

```
      ARCfile >> Arc[narc]->StartNode >> Arc[narc]->EndNode;
     }
   numarc = narc-1;

  }

void GetMiddleXY(float x1, float y1, float x2, float y2, float& xm, float& ym)
  {
   xm = (x1 + x2) / 2;
   ym = (y1 + y2) / 2;
  }

void GetXYZforArcNodes(int i, float& xs, float& ys, float& xe, float& ye,
                       short& sNd, short& eNd)
  {
   sNd = Arc[i]->StartNode;
   eNd = Arc[i]->EndNode;
   xs = Point[sNd]->x;
   ys = Point[sNd]->y;
   xe = Point[eNd]->x;
   ye = Point[eNd]->y;
  }

void GetPixelSize(int& pixsize)
  {
   cout << endl;
   cout << "Enter pixel size (in meter): " << endl;
   cin >> pixsize;
  }

void GetNumRowCol(int pixsize, float& xlength, float& ylength)
  {
   xlength = xmax - xmin;
   ylength = ymax - ymin;
   maxrow = int (ylength / pixsize) + 1;
   maxcol = ceil(int (xlength / pixsize)) + 1;
  }

void MakeRaster(int pixsize, float xlength, float ylength)
  {
   int r, c, j;
   short PointID;
   short row, col;
   double scale;
   string charX, charY, charZ;
   string charsnode, charenode;

   int narc;
   float xstart, ystart;
   float xend, yend;
   float xmid, ymid;
   short sNd, eNd;
   int numarc;

   int i;
   float dx, dy;
   float dx2, dy2;
   float x, y;
   float temp;
   float p;
```

```
float m;
bool swap;

scale = 1.0 / pixsize;

cout << endl;
cout << "xmin, ymin: " << xmin << ", " << ymin << endl;
cout << "xmax, ymax: " << xmax << ", " << ymax << endl;
cout << "pix size : " << pixsize << endl;
cout << "xlength : " << xlength << endl;
cout << "ylength : " << ylength << endl;
cout << "xmin :" << xmin << endl;
cout << "ymin :" << ymin << endl;
cout << "maxrow :" << maxrow << endl;
cout << "maxcol :" << maxcol << endl;

cout << setiosflags(ios::showpoint) << endl;
cout << setprecision(8) << "scale   : " << scale << endl;

//ifstream XYZfile("c:\\data\\lochass.xyz");
//ifstream XYZfile("c:\\data\\small.xyz");
ifstream XYZfile("c:\\data\\p2.xyz");

// reads the file header
XYZfile >> charX >> charY >> charZ;

if (! XYZfile)
    {
     cerr << "Point file not found" << endl;
     exit(1);
    }

FILE* MPDfile;
char mpdfilename[] = "c:\\data\\p2.MPD";
MPDfile = fopen(mpdfilename, "wb");

pnt = 1;

while (! XYZfile.eof())
  {
   XYZfile >> Point[npnt]->x >> Point[npnt]->y >> Point[npnt]->z;
   col = int(scale * (Point[npnt]->x - xmin)) + 1;
   row = int((-1 * scale) * (Point[npnt]->y - ymin)) + maxrow + 1;
   if (XYZfile.eof() == true)
     pnt = pnt - 1;
   PointID = pnt;
   Pixel[row][col] = PointID;
   printf("\r");
   cout << "Rasterizing the points ... " << pnt;
   pnt ++;
  }

pnt = pnt-1;
maxpnt = pnt;

GetArcFile(numarc);

for (int t = 1; t <= numarc; t ++)
  {
   GetXYZforArcNodes(t, xstart, ystart, xend, yend, sNd, eNd);
   GetMiddleXY(xstart, ystart, xend, yend, xmid, ymid);
```

```
m = (yend - ystart) / (xend - xstart);

if ((m > 0) && (m < 1))
  {
   swap = false;
   if (xstart > xend)
     {
      temp = xend;
      xend = xstart;
      xstart = temp;

      temp = yend;
      yend = ystart;
      ystart = temp;
      swap = true;
     }

   dx = xend - xstart;
   dy = yend - ystart;

   dx2 = 2 * dx;
   dy2 = 2 * dy;

   col = int(scale * (xstart - xmin)) + 1;
   row = int((-1 * scale) * (ystart - ymin)) + maxrow + 1;
   if (swap == true)
     {
      if (xstart > xmid)
         PointID = sNd;
      else
         PointID = eNd;
      Pixel[row][col] = PointID;
     }
   else
     {
      if (xstart < xmid)
         PointID = sNd;
      else
         PointID = eNd;
      Pixel[row][col] = PointID;
     }
   // initialise the error
   p = (dy2) - dx;
   xstart ++;

   while (xstart < xend)
      {
       if (p > 0)
         {
          p = p + dy2 - dx2;
          ystart ++;
         }
       else
         {
          p = p + dy2;
         }

       col = int(scale * (xstart - xmin)) + 1;
       row = int((-1 * scale) * (ystart - ymin)) + maxrow + 1;
       if (swap == true)
         {
```

```
            if (xstart > xmid)
               PointID = sNd;
            else
               PointID = eNd;
            Pixel[row][col] = PointID;
           }
         else
           {
            if (xstart < xmid)
               PointID = eNd;
            else
               PointID = sNd;
            Pixel[row][col] = PointID;
           }
         xstart ++;
        }
   }

// to handle case slope > 1
if (m > 1)
   {
    swap = false;
    if (ystart > yend)
      {
       temp = xend;
       xend = xstart;
       xstart = temp;

       temp = yend;
       yend = ystart;
       ystart = temp;
       swap = true;
      }

    dx = xend - xstart;
    dy = yend - ystart;

    dx2 = 2 * dx;
    dy2 = 2 * dy;

    col = int(scale * (xstart - xmin)) + 1;
    row = int((-1 * scale) * (ystart - ymin)) + maxrow + 1;
    if (swap == true)
      {
       if (ystart > ymid)
          PointID = sNd;
       else
          PointID = eNd;
       Pixel[row][col] = PointID;
      }
    else
      {
       if (ystart < ymid)
          PointID = sNd;
       else
          PointID = eNd;
       Pixel[row][col] = PointID;
      }
    // initialise the error
    p = (dx2) - dy;
```

```
    ystart ++;

    while (ystart < yend)
       {
        if (p > 0)
          {
           p = p + dx2 - dy2;
           xstart ++;
          }
        else
          {
           p = p + dx2;
          }

        col = int(scale * (xstart - xmin)) + 1;
        row = int((-1 * scale) * (ystart - ymin)) + maxrow + 1;
        if (swap == true)
          {
           if (ystart > ymid)
              PointID = sNd;
           else
              PointID = eNd;
           Pixel[row][col] = PointID;
          }
        else
          {
           if (ystart > ymid)  // was <
              PointID = sNd;
           else
              PointID = eNd;
           Pixel[row][col] = PointID;
          }
        ystart ++;
       }
  } // end of case slope > 1

// to handle slope m < 0 && m > -1
if ((m < 0) && (m >= -1))
  {
   swap = false;
   if (xstart > xend)
     {
      temp = xend;
      xend = xstart;
      xstart = temp;

      temp = yend;
      yend = ystart;
      ystart = temp;
      swap = true;
     }

   dx = xend - xstart;
   dy = yend - ystart;

   dx2 = 2 * dx;
   dy2 = 2 * dy;

   col = int(scale * (xstart - xmin)) + 1;
   row = int((-1 * scale) * (ystart - ymin)) + maxrow + 1;
```

```
      if (swap == true)
        {
         if (xstart > xmid)
           PointID = sNd;
         else
           PointID = eNd;
         Pixel[row][col] = PointID;
        }
      else
        {
         if (xstart < xmid)
            PointID = sNd;
         else
            PointID = eNd;

         Pixel[row][col] = PointID;
        }
      // initialise the error
      p = (dx2) + dy;
      xstart ++;

      while (xstart < xend)
         {
          if (p > 0)
            {
             p = p - (dy2 + dx2);
             ystart --;
            }
          else
            {
             p = p - dy2;
            }

          col = int(scale * (xstart - xmin)) + 1;
          row = int((-1 * scale) * (ystart - ymin)) + maxrow + 1;
          if (swap == true)
            {
             if (xstart > xmid)
                PointID = sNd;
             else
                PointID = eNd;
             Pixel[row][col] = PointID;
            }
          else
            {
             if (xstart < xmid)
                PointID = sNd;
             else
                PointID = eNd;

             Pixel[row][col] = PointID;
             }

          xstart ++;
         }

      }  // end m > -1 && m < 0

   // to handle slope m < -1
   if (m < -1)
```

```
{
 swap = false;
 if (ystart > yend)
   {
    temp = xend;
    xend = xstart;
    xstart = temp;

    temp = yend;
    yend = ystart;
    ystart = temp;
    swap = true;
   }

 dx = xend - xstart;
 dy = yend - ystart;

 dx2 = 2 * dx;
 dy2 = 2 * dy;

 col = int(scale * (xstart - xmin)) + 1;
 row = int((-1 * scale) * (ystart - ymin)) + maxrow + 1;
 if (swap == true)
   {
    if (ystart > ymid)
      PointID = sNd;
    else
      PointID = eNd;
    Pixel[row][col] = PointID;
   }
 else
   {
    if (ystart < ymid)
       PointID = sNd;
    else
       PointID = eNd;
    Pixel[row][col] = PointID;
   }

 // initialise the error
 p = dx2 + dy;
 ystart ++;

 while (ystart < yend)
    {
     if (p > 0)
       {
        p = p - (dx2 + dy2);
        xstart --;
       }
     else
       {
        p = p - dx2;
       }

     col = int(scale * (xstart - xmin)) + 1;
     row = int((-1 * scale) * (ystart - ymin)) + maxrow + 1;
     if (swap == true)
       {
        if (ystart > ymid)
```

```
                PointID = sNd;
              else
                PointID = eNd;
              Pixel[row][col] = PointID;
             }
           else
             {
              if (ystart < ymid)
                PointID = sNd;
              else
                PointID = eNd;
              Pixel[row][col] = PointID;
             }
           ystart ++;
          }
      }  // end m < -1

   // handle horizontal line
   if (ystart == yend)
      {
       while (xstart > xend)
         {
          col = int(scale * (xstart - xmin)) + 1;
          row = int((-1 * scale) * (ystart - ymin)) + maxrow + 1;
          if (xstart > xmid)
             PointID = sNd;
          else
             PointID = eNd;

          Pixel[row][col] = PointID;
          xstart = xstart - 1;
         }

          while (xstart < xend)
            {
             col = int(scale * (xstart - xmin)) + 1;
             row = int((-1 * scale) * (ystart - ymin)) + maxrow + 1;
             if (xstart < xmid)
                PointID = sNd;
             else
                PointID = eNd;

             Pixel[row][col] = PointID;
             xstart = xstart + 1;
            }

      }

   // handle vertical line
   if (xstart == xend)
      {
       while (ystart < yend)
         {
          col = int(scale * (xstart - xmin)) + 1;
          row = int((-1 * scale) * (ystart - ymin)) + maxrow + 1;
          if (ystart < ymid)
            PointID = sNd;
          else
            PointID = eNd;
```

```
            Pixel[row][col] = PointID;
            ystart = ystart + 1;
           }
         }
      } // the t loop


   // write to file
   for (row = 1; row <= maxrow+1; row ++)
     {
      fwrite(Pixel[row], sizeof(short)* maxcol, 1, MPDfile);
     }

   cout << endl;
   cout << endl;
   cout << " Total points : " << maxpnt;

   XYZfile.close();
   fclose(MPDfile);
  }

void WriteMPIfile()
  {
   FILE* MPIfile;
   char mpifilename[] = "c:\\data\\p2.MPI";
   MPIfile = fopen(mpifilename, "wb");

   // assign the values for the MPI file
   MPI.Nscanlines = maxrow + 1;
   MPI.Npixels = maxcol;
   MPI.Pvmin = 1;
   MPI.Pvmax = maxpnt;
   MPI.Maptype = 2;
   MPI.Patch = 0;
   MPI.Scale = 0;
   MPI.Crdtype = 1;
   MPI.a11 = scale;
   MPI.a12 = 0;
   MPI.a21 = 0;
   MPI.a22 = -scale;
   MPI.b1 = 1;
   MPI.b2 = maxrow + 1;

   // write out the above values in the MPI file
   fwrite(&MPI, sizeof(MpiType), 1, MPIfile);
   fclose(MPIfile);
  }


void DisplayPixel()
  {
   int i, j;
   cout << endl;
   cout << endl;
   //cout << "Displaying pixel value ..." << endl;
   cout << endl;

   for (i = 0; i <= maxrow; i ++)
     {
      //for (j = 0; j <= maxcol; j ++)
        //{
```

```
         printf("\r");
         cout << "Rasterizing row .... " << i;
         //cout << "Pixel[row][col]: Pixel[" << i << "][" << j <<
                 //"]: " << Pixel[i][j];

        //}
     }
  }

void PressAnyKey()
  {
   cout << endl;
   cout << endl;
   cout << "Press any key to continue" << endl;
   getch();
   cout << endl;
  }
```

2D Distance Transformation (DT)

The following gives the complete codes for the 2D Distance Transformation (DT).

```
//*************************************************************//
//                                                             //
// An Object-Oriented Program: Distance Transformation (DT)    //
//  Input: MPD and MPI file (rasterised points files)          //
//  Output: MPD and MPI file(DT raster files)                  //
//                                                             //
// Copyright (c) Alias Abdul-Rahman, 1999                      //
//*************************************************************//

// File: TDistanceTransform.h

#include <iostream.h>
#include <stdlib.h>
#include <stdio.h>
#include <fstream.h>
#include <io.h>
#include <iomanip.h>
#include <malloc.h>
#include <conio.h>
#include <limits.h>
#include <stddef.h>

#define masksize 8

class TDistanceTransform
  {
   public:
       // member data
       typedef struct MpiStruct
          {
           short Nscanlines;
           short Npixels;
           short Pvmin;
           short Pvmax;
           short Maptype;
           short Patch;
           short Scale;
           short Crdtype;
           float a11;
           float a12;
           float a21;
           float a22;
           float b1;
           float b2;
          } MpiType;

       typedef short Image;
```

```
    typedef Image* ImagePtr;
    typedef ImagePtr* ImagePPtr;
    typedef short Mask[9];

    MpiType MPI, MpiOut;
    Mask MaskPix;
    int row;
    int col;
    int r;
    ImagePPtr Pixel;
    int maxrow;
    int maxcol;
    FILE* MPIfile;
    FILE* MPDfile;
    FILE* MPIofile;
    FILE* MPDofile;

    // member functions
    void GetMPIInputFile();
    void GetMPDInputFile();
    void ReadImage(ImagePPtr& Pixel);
    void PrintPixel(ImagePPtr Pixel);
    void WriteMPIOutputFile();
    void WriteMPDOutputFile();
    void SetBackground(ImagePPtr Pixel, int Bg, int Fg);
    void GetUpperMask(int r, int c, ImagePPtr Pixel, Mask& MaskPix);
    void GetLowerMask(int r, int c, ImagePPtr Pixel, Mask& MaskPix);
    int MinByIndex(int from, int to);
    int Min5(int a, int b, int c, int d, int e);
    void DistancePassOne(ImagePPtr Pixel);
    void DistancePassTwo(ImagePPtr Pixel);
    void ForwardDistance();
    void BackwardDistance();
    void PressAnyKey();
    void Title();
    TDistanceTransform();         // constructor
    ~TDistanceTransform();        // destructor

};
```

```
//*************************************************************//
//                                                             //
// An Object-Oriented Program: Distance Transformation (DT)    //
//  Input: MPD and MPI file (rasterised points files)          //
//  Output: MPD and MPI file(DT raster files)                  //
// Copyright (c)Alias Abdul-Rahman, 1999                       //
//*************************************************************//

// File: TDistanceTransform.cpp

#include "TDistanceTransform.h"


void main()
  {
   TDistanceTransform DT;       // DT is an object
   DT.Title();
   TDistanceTransform();        // Allocate memory
   DT.GetMPDInputFile();
   DT.ForwardDistance();
   DT.BackwardDistance();
   DT.WriteMPDOutputFile();
   DT.WriteMPIOutputFile();
   DT.PressAnyKey();
  } // end of main


// Definitions Section

TDistanceTransform :: TDistanceTransform()         // constructor
  {
   // Allocate memory
   GetMPIInputFile();
   Pixel = new ImagePtr[maxrow];
   for (int r = 0; r < maxrow; r ++)
     {
      Pixel[r] = new Image[maxcol];
     }
  }

TDistanceTransform :: ~TDistanceTransform()        // destructor
  {
   // Deallocate memory
   for (int r = 0; r < maxrow ; r ++)
     {
      delete [] Pixel;
     }
   delete [] Pixel;
  }

void TDistanceTransform :: GetMPIInputFile()
  {
   //FILE* MPIfile;

   char mpifilename[] = "c:\\data\\pentx.mpi";
   MPIfile = fopen(mpifilename, "rb");

   if (MPIfile == NULL)
     {
       cerr << " Could not open MPI file !" << endl;
```

```
       exit(1);
      }
    else
      {
       cout << "Information of  " << mpifilename << endl;
       fread(&MPI, sizeof(MpiType), 1, MPIfile);
       int size = sizeof(MpiType);

       maxrow = MPI.Nscanlines;
       maxcol = MPI.Npixels;

       cout << "MPI file size       : " << size << endl;
       cout << endl;
       cout << "Number of lines     : " << MPI.Nscanlines << endl;
       cout << "Number of columns   : " << MPI.Npixels << endl;
       cout << "Minimum value       : " << MPI.Pvmin << endl;
       cout << "Maximum value       : " << MPI.Pvmax << endl;
       cout << "Map type            : " << MPI.Maptype << endl;
       cout << "Patched             : " << MPI.Patch << endl;
       cout << "Scale (power of 10) : " << MPI.Scale << endl;
       cout << "Coordinate type     : " << MPI.Crdtype << endl;
       cout << "Transformation parameters :" << endl;
       cout << " a11  : " << MPI.a11 << endl;
       cout << " a12  : " << MPI.a12 << endl;
       cout << " a21  : " << MPI.a21 << endl;
       cout << " a22  : " << MPI.a22 << endl;
       cout << " b1   : " << MPI.b1 << endl;
       cout << " b2   : " << MPI.b2 << endl;
       fclose(MPIfile);
      }
  }

void TDistanceTransform :: GetMPDInputFile()
  {
   char mpdifilename [] = "c:\\data\\pentx.mpd";
   MPDfile = fopen(mpdifilename, "rb");

   if (MPDfile == NULL)
     {
      cerr << "Could not open MPD file !" << endl;
      exit(1);
     }
   else
     {
      cout << endl;
      cout << "The MPD file is opened" << endl;
     }
  }

void TDistanceTransform :: ReadImage(ImagePPtr& Pixel)
  {
   cout << "Reading the data file ..." << endl;

   for (r = 0; r < maxrow; r++)
     {
      fseek(MPDfile, r * sizeof(short) * MPI.Npixels, SEEK_SET);
      fread(Pixel[r], sizeof(short) * MPI.Npixels, 1, MPDfile);
     }
   cout << endl;
   fclose(MPDfile);
  }
```

```
void TDistanceTransform :: PrintPixel(ImagePPtr Pixel)
  {
   int r, c;
   for (r = 0; r < maxrow; r ++)
     {
      for (c = 0; c < maxcol; c ++)
        {
         cout << "Pixel value at [row][col]: [" << r << "][" << c << "]: "
              << Pixel[r][c] << "\r";
        }
     }
  }

void TDistanceTransform :: WriteMPIOutputFile()
  {
   //FILE* MPIofile;
   char mpiofilename[] = "c:\\data\\pentod.MPI";
   MPIofile = fopen(mpiofilename, "w");

   MpiOut.Nscanlines = MPI.Nscanlines;
   MpiOut.Npixels = MPI.Npixels;
   MpiOut.Pvmin = 0;
   MpiOut.Pvmax = 6000;
   MpiOut.Maptype = 2;
   MpiOut.Patch = MPI.Patch;
   MpiOut.Scale = 0;
   MpiOut.Crdtype = MPI.Crdtype;
   MpiOut.a11 = 0;
   MpiOut.a12 = 0;
   MpiOut.a21 = 0;
   MpiOut.a22 = 0;
   MpiOut.b1 = 0;
   MpiOut.b2 = 0;

   fwrite(&MpiOut, sizeof(MpiType), 1, MPIofile);
   cout << endl;
   cout << "Finished writing to file." << endl;
   fclose(MPIofile);
  }

void TDistanceTransform :: WriteMPDOutputFile()
  {
   //FILE* MPDofile;
   char mpdofilename[] = "c:\\data\\pentod.mpd";
   MPDofile = fopen(mpdofilename, "w+b");
   for (r = 0; r < maxrow; r ++)
     {
      fwrite(Pixel[r], sizeof(short) * MPI.Npixels, 1, MPDofile);
     }
   fclose(MPDofile);
  }

void TDistanceTransform :: SetBackground(ImagePPtr Pixel, int Bg, int Fg)
  {
   int row;
   int col;
   cout << "SetBackground ..." << endl;
   for (row = 0; row < maxrow; row ++)
     {
      for (col = 0; col < maxcol; col ++)
```

```
        {
         if (Pixel[row][col] < 0)
             Pixel[row][col] = Bg;
         else
         if (Fg > 0)
             Pixel[row][col] = Fg;

        }
     }
  }

void TDistanceTransform :: GetUpperMask(int r, int c, ImagePPtr Pixel, Mask&
MaskPix)
  {
   MaskPix[0] = Pixel[r-1][c-1];
   MaskPix[1] = Pixel[r-1][c];
   MaskPix[2] = Pixel[r-1][c+1];
   MaskPix[3] = Pixel[r][c-1];
   MaskPix[4] = Pixel[r][c];
  }

void TDistanceTransform :: GetLowerMask(int r, int c, ImagePPtr Pixel, Mask&
MaskPix)
  {
   MaskPix[4] = Pixel[r][c];
   MaskPix[5] = Pixel[r][c+1];
   MaskPix[6] = Pixel[r+1][c-1];
   MaskPix[7] = Pixel[r+1][c];
   MaskPix[8] = Pixel[r+1][c+1];
  }

int TDistanceTransform :: MinByIndex(int from, int to)
  {
   int i;
   int temp;
   int Idx;

   temp = MaskPix[from];
   Idx = from;
   for (i = from; i <= to ; i ++)
     {
      if (temp > MaskPix[i])
        {
         temp = MaskPix[i];
         Idx = i;
        }
     }
   return Idx;
  }

int TDistanceTransform :: Min5(int a, int b, int c, int d, int e)
  {
   int mn;
   mn = min(a, b);
   mn = min(mn, c);
   mn = min(mn, d);
   mn = min(mn, e);
   return mn;
  }
```

```
void TDistanceTransform :: DistancePassOne(ImagePPtr Pixel)
  {
   int r;
   int c;
   int k;
   for (r = 1; r < maxrow; r ++)
     {
      for (c = 1; c < maxcol; c ++)
        {
         GetUpperMask(r, c, Pixel, MaskPix);
         for (k = 0; k < 4; k ++)
           {
            if ((k == 0) || (k == 2))
              MaskPix[k] = MaskPix[k] + 4;
            else
              MaskPix[k] = MaskPix[k] + 3;
           }
         if (MaskPix[4] != 30000)
             MaskPix[4] = 0;

         Pixel[r][c] = MaskPix[MinByIndex(0, 4)];

        }
      }
  }

void TDistanceTransform :: DistancePassTwo(ImagePPtr Pixel)
  {
   int r;
   int c;
   int k;

   cout << endl;
   for (r = maxrow - 2; r > 0; r --)
     {
      for (c = maxcol - 2; c > 0; c --)
        {
         GetLowerMask(r, c, Pixel, MaskPix);
         for (k = 8; k > 4; k --)
           {
            if ((k == 6) || (k == 8))
              MaskPix[k] = MaskPix[k] + 4;
            else
              MaskPix[k] = MaskPix[k] + 3;
           }
         Pixel[r][c] = MaskPix[MinByIndex(4, 8)];
        }
     }
  }

void TDistanceTransform :: ForwardDistance()
  {
   cout << "Forward Distance ..." << endl;
   ReadImage(Pixel);
   SetBackground(Pixel, 30000, 0);
   DistancePassOne(Pixel);
  }

void TDistanceTransform :: BackwardDistance()
  {
```

```
   cout << "Backward Distance ..." << endl;
   DistancePassTwo(Pixel);
  }

void TDistanceTransform :: PressAnyKey()
  {
   cout << endl;
   cout << "Press any key to continue" << endl;
   getch();
   cout << endl;
  }

void TDistanceTransform :: Title()
  {
   cout << endl;
  cout << "-----------Object-Oriented Distance Transformation --------------"
<< endl;
   cout << "----------- By: Alias Abdul-Rahman, November, 1998 -----" << endl;
   cout << " Input/Output : ILWIS .mpd, .mpi files. " << endl;
   cout << endl;
  }
```

2D Voronoi Tessellations

The following gives the complete codes for the 2D Voronoi Tessellations program.

```
//*********************************************************************//
//              A Voronoi Tesselation program for 2D TIN
//
//              Input  :  MPI and MPD files of DT program
//
//              Output :  MPI and MPD files
//
//              Copyright (c) Alias Abdul-Rahman, 1999
//
//*********************************************************************//

// File: TVoronoi.h

#include <iostream.h>
#include <stdlib.h>
#include <stdio.h>
#include <fstream.h>
#include <io.h>
#include <iomanip.h>
#include <dos.h>
#include <alloc.h>
#include <malloc.h>
#include <conio.h>
#include <limits.h>
#include <stddef.h>

#include "TDistanceTransform.h"

#define masksize 8                        // Mask size

class TVoronoiTessellation : public TDistanceTransform
  {
   public:
       TVoronoiTessellation();             // constructor
       ~TVoronoiTessellation();            // destructor

      Mask MaskPixV;
      FILE* MPDfile;
      FILE* MPDodfile;
      FILE* MPDovfile;
      FILE* MPIodfile;
      FILE* MPIovfile;
      ImagePPtr PixelV;

      // Function members
      void GetMPDInputFile();
      void ReadImage(ImagePPtr&);
      void CopyImage(ImagePPtr, ImagePPtr&);
```

```
    void GetMPIInputFile();
    void WriteMPIOutputFileDist();
    void WriteMPDOutputFileDist();
    void WriteMPIOutputFileVoronoi();
    void WriteMPDOutputFileVoronoi();
    void SetBackground(ImagePPtr, int, int);
    void GetUpperMaskDist(int, int, ImagePPtr, Mask&);
    void GetLowerMaskDist(int, int, ImagePPtr, Mask&);
    void GetUpperMaskVoronoi(int, int, ImagePPtr, Mask&);
    void GetLowerMaskVoronoi(int, int, ImagePPtr, Mask&);
    int MinByIndex(int, int);
    void ForwardPass(ImagePPtr, ImagePPtr);
    void BackwardPass(ImagePPtr, ImagePPtr);
    void ForwardVoronoi();
    void BackwardVoronoi();
    void PressAnyKey();
    void Title();
};
```

```
//*********************************************************************
//      Object-Oriented Voronoi Tesselation program for 2D TIN
//
//              Input  :  MPI and MPD files of DT program
//
//              Output :  MPI and MPD files
//
//      Copyright (c) Alias Abdul-Rahman, 1999
//
//*********************************************************************

// File: TVoronoi.cpp

#include "TVoronoi.h"

// main program
void main()
  {
   TVoronoiTessellation Voronoi;
   Voronoi.Title();
   TVoronoiTessellation();
   Voronoi.GetMPDInputFile();
   Voronoi.ForwardVoronoi();
   Voronoi.BackwardVoronoi();
   Voronoi.WriteMPDOutputFileDist();
   Voronoi.WriteMPIOutputFileDist();
   Voronoi.WriteMPDOutputFileVoronoi();
   Voronoi.WriteMPIOutputFileVoronoi();
  }

// Definition section follows:

// Constructor
TVoronoiTessellation :: TVoronoiTessellation()
 //: TDistanceTransform()
  {
   GetMPIInputFile();

   // Allocate memory dynamically for DT
   Pixel = new ImagePtr[maxrow];
   PixelV = new ImagePtr[maxrow];
   for (r = 0; r < maxrow; r ++)
     {
      Pixel[r] = new Image[maxcol];
     }

   // Allocate memory for Voronoi

   for (int j = 0; j < maxrow; j ++)
     {
      PixelV[j] = new Image[maxcol];
     }
  }

// Destructor
TVoronoiTessellation :: ~TVoronoiTessellation()
  {
   // Deallocate memory for DT
   for (r = 0; r < maxrow; r ++)
     {
```

```
      delete [] Pixel[r];
     }

   // Deallocate memory for Voronoi
   for (int j = 0; j < maxrow; j ++)
     {
      delete [] PixelV[j];
     }

   delete [] Pixel;
   delete [] PixelV;
  }

// Procedure: GetMPDFile
void TVoronoiTessellation :: GetMPDInputFile()
  {
   char mpdifilename [] = "c:\\data\\lbm.mpd";
   MPDfile = fopen(mpdifilename, "rb");

   if (MPDfile == NULL)
     {
      cerr << "Could not open MPD file !" << endl;
      exit(1);
     }
   else
     {
      cout << endl;
      cout << "The MPD file is opened" << endl;
     }
  }

// Procedure: ReadImage
void TVoronoiTessellation :: ReadImage(ImagePPtr& Pixel)
  {
   cout << "Reading the data file ..." << endl;
   int r;
   for (r = 0; r < maxrow; r++)
     {
      fseek(MPDfile, r * sizeof(short) * MPI.Npixels, SEEK_SET);
      fread(Pixel[r], sizeof(short) * MPI.Npixels, 1, MPDfile);
     }
   cout << endl;
   fclose(MPDfile);
  }

// Procedure: CopyImage
void TVoronoiTessellation :: CopyImage(ImagePPtr PixA, ImagePPtr& PixB)
  {
   int j;
   int k;
   for (j = 0; j < maxrow; j ++)
     {
     for (k = 0; k < maxcol; k ++)
       {
        PixB[j][k] = PixA[j][k];
       }
     }

  }
```

```
void TVoronoiTessellation :: GetMPIInputFile()
  {
   FILE* MPIfile;
   char mpifilename[] = "c:\\data\\lbm.mpi";
   MPIfile = fopen(mpifilename, "rb");

   if (MPIfile == NULL)
     {
       cerr << " Could not open MPI file !" << endl;
       exit(1);
     }
   else
     {
      cout << "Information of  " << mpifilename << endl;
      fread(&MPI, sizeof(MpiType), 1, MPIfile);
      int size = sizeof(MpiType);

      maxrow = MPI.Nscanlines;
      maxcol = MPI.Npixels;

      cout << "MPI file size       : " << size << endl;
      cout << endl;
      cout << "Number of lines     : " << MPI.Nscanlines << endl;
      cout << "Number of columns   : " << MPI.Npixels << endl;
      cout << "Minimum value       : " << MPI.Pvmin << endl;
      cout << "Maximum value       : " << MPI.Pvmax << endl;
      cout << "Map type            : " << MPI.Maptype << endl;
      cout << "Patched             : " << MPI.Patch << endl;
      cout << "Scale (power of 10) : " << MPI.Scale << endl;
      cout << "Coordinate type     : " << MPI.Crdtype << endl;
      cout << "Transformation parameters :" << endl;
      cout << " a11  : " << MPI.a11 << endl;
      cout << " a12  : " << MPI.a12 << endl;
      cout << " a21  : " << MPI.a21 << endl;
      cout << " a22  : " << MPI.a22 << endl;
      cout << " b1   : " << MPI.b1 << endl;
      cout << " b2   : " << MPI.b2 << endl;
      fclose(MPIfile);
     }
  }


// Procedure: WriteMPIOutputFileDist
void TVoronoiTessellation :: WriteMPIOutputFileDist()
  {
   char mpiodfilename[] = "c:\\data\\lbmtryd.MPI";
   MPIodfile = fopen(mpiodfilename, "w");

   MpiOut.Nscanlines = MPI.Nscanlines;
   MpiOut.Npixels = MPI.Npixels;
   MpiOut.Pvmin = 0;
   MpiOut.Pvmax = 6000;
   MpiOut.Maptype = 2;
   MpiOut.Patch = MPI.Patch;
   MpiOut.Scale = 0;
   MpiOut.Crdtype = MPI.Crdtype;
   MpiOut.a11 = 0;
   MpiOut.a12 = 0;
   MpiOut.a21 = 0;
   MpiOut.a22 = 0;
   MpiOut.b1 = 0;
```

```
   MpiOut.b2 = 0;

   fwrite(&MpiOut, sizeof(MpiType), 1, MPIodfile);
   cout << endl;
   cout << "Finished writing to file." << endl;
   fclose(MPIodfile);
  }


// Procedure: WriteMPDOutputFileDist
void TVoronoiTessellation :: WriteMPDOutputFileDist()
  {
   int j;
   char mpdodfilename[] = "c:\\data\\lbmtryd.MPD";
   MPDodfile = fopen(mpdodfilename, "w+b");
   for (j = 0; j < maxrow; j ++)
     {
      fwrite(Pixel[j], sizeof(short) * MPI.Npixels, 1, MPDodfile);
     }
   fclose(MPDodfile);
  }


// Procedure: WriteMPIOutputVoronoiFile
void TVoronoiTessellation :: WriteMPIOutputFileVoronoi()
  {
   char mpiovfilename[] = "c:\\data\\lbmtryv.MPI";
   MPIovfile = fopen(mpiovfilename, "w");

   MpiOut.Nscanlines = MPI.Nscanlines;
   MpiOut.Npixels = MPI.Npixels;
   MpiOut.Pvmin = 1;
   MpiOut.Pvmax = 7000;
   MpiOut.Maptype = 2;
   MpiOut.Patch = MPI.Patch;
   MpiOut.Scale = 0;
   MpiOut.Crdtype = MPI.Crdtype;
   MpiOut.a11 = 0;
   MpiOut.a12 = 0;
   MpiOut.a21 = 0;
   MpiOut.a22 = 0;
   MpiOut.b1 = 0;
   MpiOut.b2 = 0;

   fwrite(&MpiOut, sizeof(MpiType), 1, MPIovfile);
   cout << endl;
   cout << "Finished writing to file." << endl;
   fclose(MPIovfile);
  }


// Procedure: WriteMPDOutputFileVoronoi
void TVoronoiTessellation :: WriteMPDOutputFileVoronoi()
  {
   int j;
   char mpdovfilename[] = "c:\\data\\lbmtryv.MPD";
   MPDovfile = fopen(mpdovfilename, "w+b");
   for (j = 0; j < maxrow; j ++)
     {
      fwrite(PixelV[j], sizeof(short) * MPI.Npixels, 1, MPDovfile);
     }
```

```
   fclose(MPDovfile);
  }

// Procedure: SetBackground
void TVoronoiTessellation :: SetBackground(ImagePPtr Pixel, int Bg, int Fg)
  {
   int row;
   int col;
   for (row = 0; row < maxrow; row ++)
     {
      for (col = 0; col < maxcol; col ++)
        {
         if (Pixel[row][col] <= 0)
             Pixel[row][col] = Bg;
         else
         if (Fg > 0)
             Pixel[row][col] = Fg;

        }
     }
  }

// Procedure: GetUpperMaskDist
void TVoronoiTessellation :: GetUpperMaskDist(int r, int c, ImagePPtr Pixel,
Mask& MaskPix)
  {
   MaskPix[0] = Pixel[r-1][c-1];
   MaskPix[1] = Pixel[r-1][c];
   MaskPix[2] = Pixel[r-1][c+1];
   MaskPix[3] = Pixel[r][c-1];
   MaskPix[4] = Pixel[r][c];
  }

// Procedure: GetLowerMaskDist
void TVoronoiTessellation :: GetLowerMaskDist(int r, int c, ImagePPtr Pixel,
Mask& MaskPix)
  {
   MaskPix[4] = Pixel[r][c];
   MaskPix[5] = Pixel[r][c+1];
   MaskPix[6] = Pixel[r+1][c-1];
   MaskPix[7] = Pixel[r+1][c];
   MaskPix[8] = Pixel[r+1][c+1];
  }

// Procedure: GetUpperMaskVoronoi
void TVoronoiTessellation :: GetUpperMaskVoronoi(int r, int c, ImagePPtr
PixelV, Mask& MaskPixV)
  {
   MaskPixV[0] = PixelV[r-1][c-1];
   MaskPixV[1] = PixelV[r-1][c];
   MaskPixV[2] = PixelV[r-1][c+1];
   MaskPixV[3] = PixelV[r][c-1];
   MaskPixV[4] = PixelV[r][c];
  }

// Procedure: GetLowerMaskVoronoi
void TVoronoiTessellation :: GetLowerMaskVoronoi(int r, int c, ImagePPtr
PixelV, Mask& MaskPixV)
  {
   MaskPixV[4] = PixelV[r][c];
   MaskPixV[5] = PixelV[r][c+1];
```

```
   MaskPixV[6] = PixelV[r+1][c-1];
   MaskPixV[7] = PixelV[r+1][c];
   MaskPixV[8] = PixelV[r+1][c+1];
  }

// Function: MinByIndex
int TVoronoiTessellation :: MinByIndex(int from, int to)
  {
   int i;
   int temp;
   int Idx;

   temp = MaskPix[from];
   Idx = from;
   for (i = from; i <= to ; i ++)
     {
      if (temp > MaskPix[i])
        {
         temp = MaskPix[i];
         Idx = i;
        }
     }
   return Idx;
  }

// Procedure: ForwardPass
void TVoronoiTessellation :: ForwardPass(ImagePPtr Pixel, ImagePPtr PixelV)
  {
   int r;
   int c;
   int k;
   for (r = 1; r < maxrow; r ++)
     {

      for (c = 1; c < maxcol; c ++)
        {
         GetUpperMaskDist(r, c, Pixel, MaskPix);
         GetUpperMaskVoronoi(r, c, PixelV, MaskPixV);
         for (k = 0; k < 4; k ++)
           {
            if ((k == 0) || (k == 2))
              MaskPix[k] = MaskPix[k] + 4;
            else
              MaskPix[k] = MaskPix[k] + 3;
           }
         if (MaskPix[4] != 255)
             MaskPix[4] = 0;

         Pixel[r][c] = MaskPix[MinByIndex(0, 4)];
         PixelV[r][c] = MaskPixV[MinByIndex(0, 4)];
       }
     }

  }

// Procedure: BackwardPass
void TVoronoiTessellation :: BackwardPass(ImagePPtr Pixel, ImagePPtr PixelV)
  {
   int r;
   int c;
```

```
   int k;

   cout << endl;
   for (r = maxrow - 2; r > 0; r --)
     {

      for (c = maxcol - 2; c > 0; c --)
        {
         GetLowerMaskDist(r, c, Pixel, MaskPix);
         GetLowerMaskVoronoi(r, c, PixelV, MaskPixV);
         for (k = 8; k > 4; k --)
           {
            if ((k == 6) || (k == 8))
              MaskPix[k] = MaskPix[k] + 4;
            else
              MaskPix[k] = MaskPix[k] + 3;
           }
         Pixel[r][c] = MaskPix[MinByIndex(4, 8)];
         PixelV[r][c] = MaskPixV[MinByIndex(4, 8)];
        }
     }
  }

// Procedure: ForwardVoronoi
void TVoronoiTessellation :: ForwardVoronoi()
  {
   cout << "Forward pass ..." << endl;
   ReadImage(Pixel);
   CopyImage(Pixel, PixelV);
   SetBackground(Pixel, 255, 0);
   ForwardPass(Pixel, PixelV);

  }

// Procedure: BackwardVoronoi
void TVoronoiTessellation :: BackwardVoronoi()
  {
   cout << "Backward pass ..." << endl;
   BackwardPass(Pixel, PixelV);

  }

// Procedure: PressAnyKey
void TVoronoiTessellation :: PressAnyKey()
  {
   cout << endl;
   cout << "Press any key to continue" << endl;
   getch();
   cout << endl;
  }


// Procedure: Title
void TVoronoiTessellation :: Title()
  {
   cout << endl;
    cout << "-----------Object-Oriented Distance Transformation-----------"
      << endl;
   cout << "-----------          and Voronoi Tessellation      -----------"
      << endl;
    cout << "----------- By: Alias Abdul-Rahman, November, 1998 -----" <<
```

```
    endl;
 cout << " Input/Output : ILWIS .mpd, .mpi files. " << endl;
 cout << endl;

 cout << endl;
 char anykey;
 cout << "Press any key to continue." << endl;
 getch();
}
```

2DTIN Data Structuring Code

The following gives the complete code for the 2D TIN data structure generation program.

```
//*************************************************************************//
//                Object-Oriented Voronoi-to-TIN program
//
//               Input  :  MPI and MPD file of DT program
//
//               Output :  MPI and MPD files
//
//
//              Copyright (c) Alias Abdul-Rahman, 1999
//*************************************************************************//

#include <iostream.h>
#include <stdlib.h>
#include <stdio.h>
#include <fstream.h>
#include <io.h>
#include <iomanip.h>
#include <dos.h>
#include <alloc.h>
#include <malloc.h>
#include <conio.h>
#include <limits.h>
#include <stddef.h>

#include "TVoronoi.h"
//#include "TDistanceTransform.h"

// Constant section

#define masksize4 4               // 2 x 2 Mask size

class TTinGeneration : public TVoronoiTessellation
  {
   public:
      TTinGeneration();           // constructor
      ~TTinGeneration();          // destructor

      typedef short DataType;
      typedef struct VertexStruct
                 {
                  DataType N1;
                  DataType N2;
                  DataType N3;
                 } TVertex;
```

```
    typedef struct TsNodeStruct
              {
               short x;
               short y;
              } TsNode;

    // data members
    typedef short Image;
    typedef Image* ImagePtr;
    typedef ImagePtr* ImagePPtr;
    typedef short Mask[4];

    //MpiType MPI;
    Mask MaskPix;
    ImagePPtr Pixel;
    FILE* MPDfile;
    FILE* MPIfile;

    FILE* TINfile;
    int i, r;
    int NTri;
    TVertex Element;
    TsNode TsPnt;
    bool FoundTri;

    // function members
    void GetMPDfile();
    void ReadImage(ImagePPtr&);
    void GetMPIfile();
    void GetTINfile();
    bool Less(DataType, DataType);
    bool Greater(DataType, DataType);
    void Swap(DataType&, DataType&);
    void NodeOrder(DataType&, DataType&, DataType&);
    void AddTritoFile(TVertex);
    void GetMask(int, int, ImagePPtr, Mask&);
    void GetSubImage(int, int, ImagePPtr, Mask&);
    void ScanlinesUp(Mask);
    void ScanlinesDown(Mask);
    void Scanlines(ImagePPtr);
    void MakeTIN();
    void Title();
    void PressAnyKey();
};
```

```
// Object-Oriented Program for Generating TIN
// File: TTinGen.cpp

#include "TTinGen.h"

// main program
void main()
  {
   TTinGeneration TIN;    // TIN is an object
   TIN.Title();
   TTinGeneration();      // constructor
   TIN.GetMPDfile();
   TIN.GetTINfile();
   TIN.MakeTIN();
   TIN.PressAnyKey();
  }


// Definitions section follows:
// A constructor
TTinGeneration :: TTinGeneration()
  {
   GetMPIfile();

   // Allocate memory dynamically
   Pixel = new ImagePtr[maxrow];
   for (r = 0; r < maxrow; r ++)
     {
      Pixel[r] = new Image[maxcol];
     }
  }

// A destructor
TTinGeneration :: ~TTinGeneration()
  {
   // Deallocate memory
   for (r = 0; r < maxrow; r ++)
     {
      delete [] Pixel[r];
     }

   delete [] Pixel;
  }


// Procedure: PresAnyKey
void TTinGeneration :: PressAnyKey()
  {
   cout << endl;
   cout << "Press any key to continue" << endl;
   getch();
   cout << endl;
  }

// Procedure: GetMPDFile
void TTinGeneration :: GetMPDfile()
  {
   char mpdfilename [] = "c:\\data\\tryserv.mpd";
   MPDfile = fopen(mpdfilename, "rb");
```

```
   if (MPDfile == NULL)
     {
      cerr << "Could not open MPD file !" << endl;
      exit(1);
     }
   else
     {
      cout << endl;
      cout << "The MPD file is opened" << endl;
     }
  }

void TTinGeneration :: ReadImage(ImagePPtr& Pixel)
  {
   int r;
   for (r = 0; r < maxrow; r++)
     {
      fseek(MPDfile, r * sizeof(short) * MPI.Npixels, SEEK_SET);
      fread(Pixel[r], sizeof(short) * MPI.Npixels, 1, MPDfile);
     }
   cout << endl;
   fclose(MPDfile);
  }

void TTinGeneration :: GetMPIfile()
  {
   char mpifilename [] = "c:\\data\\tryserv.mpi";
   MPIfile = fopen(mpifilename, "rb");

   if (MPIfile == NULL)
     {
      cerr << "Could not open MPD file !" << endl;
      exit(1);
     }
   else
     {
      cout << "Information of  " << mpifilename << endl;
      fread(&MPI, sizeof(MpiType), 1, MPIfile);
      int size = sizeof(MpiType);

      maxrow = MPI.Nscanlines;
      maxcol = MPI.Npixels;

      cout << "MPI file size       : " << size << endl;
      cout << endl;
      cout << "Number of lines     : " << MPI.Nscanlines << endl;
      cout << "Number of columns   : " << MPI.Npixels << endl;
      cout << "Minimum value       : " << MPI.Pvmin << endl;
      cout << "Maximum value       : " << MPI.Pvmax << endl;
      cout << "Map type            : " << MPI.Maptype << endl;
      cout << "Patched             : " << MPI.Patch << endl;
      cout << "Scale (power of 10) : " << MPI.Scale << endl;
      cout << "Coordinate tyoe     : " << MPI.Crdtype << endl;
      cout << "Transformation parameters :" << endl;
      cout << " a11  : " << MPI.a11 << endl;
      cout << " a12  : " << MPI.a12 << endl;
      cout << " a21  : " << MPI.a21 << endl;
      cout << " a22  : " << MPI.a22 << endl;
      cout << " b1   : " << MPI.b1 << endl;
      cout << " b2   : " << MPI.b2 << endl;
      fclose(MPIfile);
```

```
    }
  }

// Procedure: GetTINfile
void TTinGeneration :: GetTINfile()
  {
   char TINfilename [] = "c:\\data\\tryser.tin";
   TINfile = fopen(TINfilename, "w");

   char* charN1 = "Node1";
   char* charN2 = "Node2";
   char* charN3 = "Node3";

   fprintf(TINfile, "%8s %8s %8s \n", charN1, charN2, charN3);
   if (TINfile == NULL)
     {
      cerr << "Could not open TIN file !" << endl;
      exit(1);
     }
   else
     {
      cout << endl;
      cout << "The TIN file is okay for writing ... " << endl;
     }
   //fclose(TINfile);
  }

// Function: Less
bool TTinGeneration :: Less(DataType a, DataType b)
  {
   return (a < b);
  }

// Function: Greater
bool TTinGeneration :: Greater(DataType a, DataType b)
  {
   return (a > b);
  }

//Procedure: Swap
void TTinGeneration :: Swap(DataType& a, DataType& b)
  {
   DataType temp;

   temp = a;
   a = b;
   b = temp;
  }

// Procedure: NodeOrder
void TTinGeneration :: NodeOrder(DataType& a, DataType& b, DataType& c)
  {
   if (Greater(a, b))
       Swap(a, b);
   if (Greater(b, c))
       Swap(b, c);
   if (Greater(a, b))
       Swap(a, b);
  }
```

```
// Procedure: AddTritoFile
void TTinGeneration :: AddTritoFile(TVertex Triangle)
  {
   fprintf(TINfile, "%8d %8d %8d \n", Triangle.N1, Triangle.N2, Triangle.N3);
  }

//Procedure: GetSubImage
void TTinGeneration :: GetSubImage(int i, int c, ImagePPtr Pixel, Mask&
MaskPix)
  {
   int m, j;
   m = -1;
   for (i = 0; i < maxrow; i ++)
     {
      for (j = c; j < c + 1; j ++)
        {
         m ++;
         MaskPix[m] = Pixel[i][j];
        }
     }
  }

// Procedure: GetMask
void TTinGeneration :: GetMask(int r, int c, ImagePPtr Pixel, Mask& MaskPix)
  {
   MaskPix[0] = Pixel[r][c];
   MaskPix[1] = Pixel[r][c+1];
   MaskPix[2] = Pixel[r+1][c];
   MaskPix[3] = Pixel[r+1][c+1];
  }

// Procedure: ScanlinesUp
void TTinGeneration :: ScanlinesUp(Mask MaskPix)
  {
   bool DetectTri;
   bool NarrowBand;

   DetectTri = (MaskPix[0] != MaskPix[1]) &&
               (MaskPix[1] != MaskPix[2]) &&
               (MaskPix[2] != MaskPix[0]);

   NarrowBand = (MaskPix[0] == MaskPix[3]);
   NTri = 0;
   if (DetectTri && (! NarrowBand))
     {
      Element.N1 = MaskPix[0];
      Element.N2 = MaskPix[1];
      Element.N3 = MaskPix[2];
      NodeOrder(Element.N1, Element.N2, Element.N3);
      //NTri ++;
      AddTritoFile(Element);
      NTri ++;
      FoundTri = true;
     }
   else
     FoundTri = false;
  }

// Procedure: ScanlinesDown
void TTinGeneration :: ScanlinesDown(Mask MaskPix)
  {
```

```
   bool DetectTri;
   bool NarrowBand;

   DetectTri = (MaskPix[1] != MaskPix[2]) &&
               (MaskPix[2] != MaskPix[3]) &&
               (MaskPix[3] != MaskPix[1]);

   NarrowBand = (MaskPix[0] == MaskPix[3]);

   NTri = 0;
   if (DetectTri && (! NarrowBand))
      {
       Element.N1 = MaskPix[1];
       Element.N2 = MaskPix[2];
       Element.N3 = MaskPix[3];
       NodeOrder(Element.N1, Element.N2, Element.N3);
       //NTri ++;
       AddTritoFile(Element);
       NTri ++;
       FoundTri = true;
      }
   else
      FoundTri = false;
  }

// Procedure: Scanlines
void TTinGeneration :: Scanlines(ImagePPtr Pixel)
  {
   int r, c;
   for (r = 1; r < maxrow-1; r ++)
     {
      for (c = 1; c < maxcol-1; c ++)
        {
         GetMask(r, c, Pixel, MaskPix);
         ScanlinesUp(MaskPix);
         ScanlinesDown(MaskPix);
        }
     }
  }

// Procedure: MakeTIN
void TTinGeneration :: MakeTIN()
  {
   cout << "Reading the data ..." << endl;
   ReadImage(Pixel);
   cout << endl;
   cout << "Generating TIN ..." << endl;
   Scanlines(Pixel);
   cout << endl;
   cout << "Writing TIN output to file ... " << endl;
   fclose(TINfile);
  }

//Procedure: Title
void TTinGeneration :: Title()
  {
   cout << endl;
           cout    <<    "--------------    Object-Oriented    Voronoi-to-TIN
-----------------------" << endl;
  cout << "----------- By: Alias Abdul-Rahman, November, 1998 --------------"
<< endl;
```

```
 cout << "           Input : ILWIS .mpd, .mpi files. " << endl;
 cout << "           Output: TIN file (.tin), an ASCII file. " << endl;
 cout << endl;
}
```

```
// tinwin2d.rh

#define IDI_ICON1 1
#define CM_CLEAR           1125
#define CM_FILEEXIT        1126
#define CM_ABOUT           1127
#define CM_INPUTFILESITEM1    18874 // for XYZ file
#define CM_POPUPITEM1      18873 // for TIN file
```

2D Viewing Code

The following gives the complete code for the 2D viewing (SDI) in Object Windows Library (OWL) environment.

```
//******************************************************//
// A Windows program for TIN display                    //
//   Input: XYZ coordinates and TIN file                //
//   Output: TINs in Windows environment                //
//   Copyright (c) Alias Abdul-Rahman , 1999            //
//******************************************************//

#include <owl/pch.h>
#include <owl/applicat.h>
#include <owl/framewin.h>
#include <owl/decframe.h>
#include <owl/statusba.h>
#include <owl/controlb.h>
#include <owl/buttonga.h>
#include <owl/gdiobjec.h>
#include <owl/chooseco.h>
#include <owl/inputdia.h>
#include <owl/opensave.h>
#include <owl/dc.h>
#include <mem.h>

#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <stdio.h>
#include <stddef.h>
#include <string.h>

#include "tinwin2d.rh"

#define maxtriangle 20000
#define maxpoint 8000
#define maxarc 200

class TTINDrawApp : public TApplication
  {
   public:
      TTINDrawApp() : TApplication() {}  // call base class constructor
      void InitMainWindow();
  };


class TTINWindow : public TWindow
  {
   public:
      TTINWindow(TWindow* parent = 0)
```

```
        : TWindow(parent) {}

      struct Point
        {
         float x;
         float y;
         float z;
        };

      struct Triangle
        {
         int Node1;
         int Node2;
         int Node3;
        };

      Point* pnt_ptr[maxpoint];
      Triangle* tri_ptr[maxtriangle];

      struct Polygon
        {
         int Snode;
         int Enode;
        };

      Polygon* poly_ptr[maxarc];

      FILE* XYZfile;
      FILE* TINfile;
      FILE* ARCfile;

      TOpenSaveDialog :: TData fileData;
      int k, i, t, ii;
      float xmin, xmax, ymin, ymax;
      float zmin, zmax;
      float xlength, ylength;
      int npnt, ntri, narc;
      int MaxX, MaxY;
      float scaleX, scaleY;

      void ReadXYZ();
      void ReadTIN();
      void ReadARC();
      void GetMinMax(float&, float&,
                     float&, float&,
                     float&, float&);
      void Paint(TDC& dc, bool, TRect&);
      void EvSize(uint, TSize&);
      void CmInputFilesItem1();
      void CmPopupItem1();
      void CmFileExit()
           {PostQuitMessage(0);}
      void CmClear();
      void CmAbout();
      DECLARE_RESPONSE_TABLE(TTINWindow);

  };

DEFINE_RESPONSE_TABLE1(TTINWindow, TWindow)
  EV_WM_SIZE,
```

```
  EV_COMMAND(CM_INPUTFILESITEM1, CmInputFilesItem1),
  EV_COMMAND(CM_POPUPITEM1, CmPopupItem1),
  EV_COMMAND(CM_ABOUT, CmAbout),
  EV_COMMAND(CM_FILEEXIT, CmFileExit),
  EV_COMMAND(CM_CLEAR, CmClear),
END_RESPONSE_TABLE;


void TTINWindow :: CmInputFilesItem1()
  {
   ReadXYZ();
  }

void TTINWindow :: CmPopupItem1()
  {
   ReadTIN();
  }

void TTINWindow :: ReadXYZ()
  {
   string charX, charY, charZ;

   // Allocate memory for XYZ points
   for (i = 0; i < maxpoint; i ++)
     {
      pnt_ptr[i] = new Point;
     }

   // Reads again the the XYZ file
   //ifstream XYZfile = "c:\\data\\lochgrd.xyz";
   //ifstream XYZfile = "c:\\data\\lochass.xyz";
   //ifstream XYZfile = "c:\\data\\ard.xyz";
   ifstream XYZfile = "c:\\data\\pent.xyz";

   if (! XYZfile)
     {
      MessageBox ("Unable to open file: XYZ file", "FileError",
                  MB_OK | MB_ICONEXCLAMATION);
     }
   else
     {
      // reads the file header
      XYZfile >> charX >> charY >> charZ;
      npnt = 0;
        do
         {
          npnt ++;
          XYZfile >> pnt_ptr[npnt] -> x
                  >> pnt_ptr[npnt] -> y
                  >> pnt_ptr[npnt] -> z;

         } while (! XYZfile.eof());

     //if (XYZfile.eof() == true)
        //MessageBox ("Finished read the XYZ points", "XYZ file complete",
                    //MB_OK | MB_ICONINFORMATION);
     }
   XYZfile.close();
  }

void TTINWindow :: ReadTIN()
```

```
  {
   string charNode1, charNode2, charNode3;

   // Allocate memory for Triangle
   for (t = 0; t < maxtriangle; t ++)
     {
      tri_ptr[t] = new Triangle;
     }

   // Reads the TIN file
   //ifstream TINfile = "c:\\data\\lochgrd.tin";
   //ifstream TINfile = "c:\\data\\lochas2.tin";
   //ifstream TINfile = "c:\\data\\ard.tin";
   ifstream TINfile = "c:\\data\\pent.tin";


   if (!TINfile)
     {
      MessageBox("Unable to open file: TIN file",
                 "File Error", MB_OK | MB_ICONEXCLAMATION);
     }

   else
     {
      ntri = 0;

      // reads the file header
      TINfile >> charNode1 >> charNode2 >> charNode3;
      do
        {
         ntri ++;
         TINfile >> tri_ptr[ntri] -> Node1
                 >> tri_ptr[ntri] -> Node2
                 >> tri_ptr[ntri] -> Node3;

        } while (! TINfile.eof());

       //if (TINfile.eof() == true)
        //MessageBox ("Finished read the TIN nodes", "TIN file complete",
                    //MB_OK | MB_ICONINFORMATION);
      }

   TINfile.close();
  }

void TTINWindow :: ReadARC()
  {
   string charSnode, charEnode;

   // Allocate memory for XYZ points
   for (int ii = 0; ii < maxarc; ii ++)
     {
      poly_ptr[ii] = new Polygon;
     }

   // Reads again the the XYZ file
   ifstream ARCfile = "c:\\data\\lake.arc";


   if (! ARCfile)
     {
```

```
      MessageBox ("Unable to open file: ARC file", "FileError",
                  MB_OK | MB_ICONEXCLAMATION);
     }
   else
     {
      // reads the file header
      ARCfile >> charSnode >> charEnode;
      narc = 0;
        do
         {
          narc ++;
          ARCfile >> poly_ptr[narc] -> Snode
                  >> poly_ptr[narc] -> Enode;

         } while (! ARCfile.eof());

     //if (ARCfile.eof() == true)
        //MessageBox ("Finished read the ARC points", "ARC file complete",
                  //MB_OK | MB_ICONINFORMATION);
     }
   ARCfile.close();
  }


void TTINWindow :: CmClear()
  {
   // to clear and redraw
   Invalidate();
  }


void TTINWindow :: CmAbout()
  {
   MessageBox ("TINs display with menu - by Alias Abdul-Rahman, 1999",
               "About the Program", MB_OK | MB_ICONINFORMATION);
  }

void TTINWindow :: GetMinMax(float& xmax, float& ymax,
                             float& xmin, float& ymin,
                             float& xlength, float& ylength)
  {
   string charX, charY, charZ;

   // Read the XYZ file and get the min-max of the coordinates
   //ifstream XYZfile = "c:\\data\\lochgrd.xyz";
   //ifstream XYZfile = "c:\\data\\lochass.xyz";
   //ifstream XYZfile = "c:\\data\\ard.xyz";
   ifstream XYZfile = "c:\\data\\pent.xyz";

   if (! XYZfile)
      MessageBox ("Unable to open file: XYZ file", "FileError",
                  MB_OK | MB_ICONEXCLAMATION);

   float x, y, z;

   // reads the file header
   XYZfile >> charX >> charY >> charZ;

   XYZfile >> x >> y >> z;
      xmin = x;
      xmax = x;
```

```
      ymin = y;
      ymax = y;

      zmin = z;
      zmax = z;

      do
        {
         XYZfile >> x >> y >> z;

         if (xmin > x)
             xmin = x;

         if (ymin > y)
             ymin = y;

         if (zmin > z)
             zmin = z;

         if (xmax < x)
             xmax = x;

         if (ymax < y)
             ymax = y;

         if (zmax < z)
             zmax = z;
         } while (! XYZfile.eof());

      xlength = xmax - xmin;
      ylength = ymax - ymin;
  }

void TTINWindow :: Paint(TDC& dc, bool, TRect&)
  {
   GetMinMax(xmax, ymax, xmin, ymin, xlength, ylength);

   ReadXYZ(); // reads the XYZ
   ReadTIN(); // reads the TIN
   //ReadARC(); // reads the ARC

   // to set the background color of the client area
   // comment out the next line for default colour
   //static const TColor color (RGB(0, 0, 0));      // RGB (0, 0, 0) - black
   //TWindow :: SetBkgndColor(color);               // RGB (192, 192, 192) -
gray


   RECT rect;
   :: GetClientRect(HWindow, &rect);

   dc.SetMapMode(MM_ANISOTROPIC);

   scaleX = (rect.right - rect.left) / xlength;
   scaleY = (rect.bottom - rect.top) / ylength;

   for (k = 1; k < npnt; k ++)
     {
      pnt_ptr[k] -> x = (pnt_ptr[k] -> x - xmin) * scaleX;
      pnt_ptr[k] -> y = (rect.bottom - rect.top) - ((pnt_ptr[k] -> y - ymin)
                                                    * scaleY);
```

```
  }

// set pen for points display
TPen pen1 = TPen(RGB(255, 0, 0), 1, PS_SOLID);
SelectObject(dc, pen1);
for (k = 1; k < npnt; k ++)
  {
   dc.MoveTo(pnt_ptr[k] -> x, pnt_ptr[k] -> y);
   dc.LineTo(pnt_ptr[k] -> x, pnt_ptr[k] -> y);
  }

SelectObject(dc, pen1);
DeleteObject(pen1);


// set pen style for TINs display
TPen pen2 = TPen(RGB(0, 0, 255), 1, PS_SOLID);
SelectObject(dc, pen2);

for (k = 1; k < ntri; k ++)
  {
   dc.MoveTo(pnt_ptr[tri_ptr[k] -> Node1] -> x,
             pnt_ptr[tri_ptr[k] -> Node1] -> y);

   dc.LineTo(pnt_ptr[tri_ptr[k] -> Node2] -> x,
             pnt_ptr[tri_ptr[k] -> Node2] -> y);
   dc.LineTo(pnt_ptr[tri_ptr[k] -> Node3] -> x,
             pnt_ptr[tri_ptr[k] -> Node3] -> y);
   dc.LineTo(pnt_ptr[tri_ptr[k] -> Node1] -> x,
             pnt_ptr[tri_ptr[k] -> Node1] -> y);
  }
SelectObject(dc, pen2);
DeleteObject(pen2);



// set pen style for polygon (i.e. constrained edges) display
TPen pen3 = TPen(RGB(255, 0, 0), 2, PS_SOLID);
SelectObject(dc, pen3);

for (int r = 1; r < narc; r ++)
  {
   dc.MoveTo(pnt_ptr[poly_ptr[r] -> Snode] -> x,
             pnt_ptr[poly_ptr[r] -> Snode] -> y);
   dc.LineTo(pnt_ptr[poly_ptr[r] -> Enode] -> x,
             pnt_ptr[poly_ptr[r] -> Enode] -> y);
   dc.LineTo(pnt_ptr[poly_ptr[r] -> Snode] -> x,
             pnt_ptr[poly_ptr[r] -> Snode] -> y);
  }
SelectObject(dc, pen3);
DeleteObject(pen3);



// Deallocate memory
for (i = 0; i < maxpoint; i ++)
  {
   delete [] pnt_ptr[i];
  }

for (t = 0; t < maxtriangle; t ++)
```

```
      {
       delete [] tri_ptr[t];
      }

   for (ii = 0; ii < maxarc; ii ++)
      {
       delete [] poly_ptr[ii];
      }

  }

// Force the window to repaint if resize
void TTINWindow :: EvSize(uint, TSize&)
  {
   :: InvalidateRect(HWindow, 0, true);
  }

void TTINDrawApp :: InitMainWindow()
  {
   // construct the decorated MDI frame window
   TDecoratedFrame* frame = new TDecoratedFrame(0,
                      "TINSoft ver. 1.0", new TTINWindow);

   // construct the status bar
   TStatusBar* sb = new TStatusBar(frame, TGadget :: Recessed,
                                   TStatusBar :: CapsLock | TStatusBar ::
NumLock);

   // construct the control bar
   TControlBar* cb = new TControlBar(frame);

   cb -> Insert(*new TButtonGadget(CM_INPUTFILESITEM1, CM_INPUTFILESITEM1,
                      TButtonGadget :: Command));
   cb -> Insert(*new TButtonGadget(CM_POPUPITEM1, CM_POPUPITEM1,
                      TButtonGadget :: Command));

   cb -> Insert(*new TSeparatorGadget);

   cb -> Insert(*new TButtonGadget(CM_CLEAR, CM_CLEAR,
                      TButtonGadget :: Command));

   cb -> Insert(*new TSeparatorGadget);

   cb -> Insert(*new TButtonGadget(CM_FILEEXIT, CM_FILEEXIT,
                      TButtonGadget :: Command));

   cb -> Insert(*new TSeparatorGadget);

   cb -> Insert(*new TButtonGadget(CM_ABOUT, CM_ABOUT,
                      TButtonGadget :: Command));

   cb -> SetHintMode(TGadgetWindow :: EnterHints);

   // set client area to the application workspace
   frame -> SetBkgndColor(::GetSysColor(COLOR_APPWORKSPACE));

   // insert the status and control bar into the frame
   frame -> Insert(*sb, TDecoratedFrame :: Bottom);
   frame -> Insert(*cb, TDecoratedFrame :: Top);

   // set the main window and its menu
```

```
   SetMainWindow(frame);
   GetMainWindow() -> AssignMenu("COMMANDS");
   EnableCtl3d(true);
  }


int OwlMain(int /*argc*/, char* /*argv*/ [])
  {
   return TTINDrawApp().Run();
  }
```

```
#include "tinwin2d.rh"

COMMANDS MENU
{
 POPUP "&InputFiles"
 {
  MENUITEM "&XYZ File", CM_INPUTFILESITEM1
  MENUITEM "&TIN File", CM_POPUPITEM1
 }

 POPUP "&Redraw"
 {
  MENUITEM "&Clear and Redraw", CM_CLEAR
  MENUITEM "E&xit", CM_FILEEXIT
 }

 MENUITEM "&About", CM_ABOUT
}

STRINGTABLE
  {
   CM_INPUTFILESITEM1,  "Get the XYZ file"
   CM_POPUPITEM1,       "Get the TIN file"
   CM_CLEAR,            "Redraw the TINs"
   CM_FILEEXIT,         "Exit the program ..."
   CM_ABOUT,            "About the program ..."
  }

CM_CLEAR BITMAP
{
 '42 4D 66 01 00 00 00 00 00 00 76 00 00 00 28 00'
 '00 00 14 00 00 00 14 00 00 00 01 00 04 00 00 00'
 '00 00 F0 00 00 00 00 00 00 00 00 00 00 00 00 00'
 '00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 80'
 '00 00 00 80 80 00 80 00 00 00 80 00 80 00 80 80'
 '00 00 C0 C0 C0 00 80 80 80 00 00 00 FF 00 00 FF'
 '00 00 00 FF FF 00 FF 00 00 00 FF 00 FF 00 FF FF'
 '00 00 FF FF FF 00 91 77 77 77 77 77 77 77 77 99'
 '00 00 79 77 77 77 77 77 77 77 79 17 00 00 77 17'
 '77 87 77 77 77 79 91 77 00 00 77 77 80 08 88 87'
 '77 77 77 77 00 00 77 78 00 00 08 88 77 77 77 77'
 '00 00 77 78 0A 0A A0 08 88 87 77 77 00 00 77 78'
 '0A 0A AA A0 88 88 77 77 00 00 77 80 AA 0A AA AA'
 '00 88 77 77 00 00 77 80 AA A0 AA A0 A0 88 77 77'
 '00 00 77 0A AA A0 AA 0A A0 88 71 17 00 00 77 0A'
 'AA A0 00 AA A0 88 79 97 00 00 78 0A AA A0 AA AA'
 'A0 88 77 77 00 00 78 00 00 00 AA AA A0 88 77 77'
 '00 00 78 88 87 78 0A AA A0 88 77 77 00 00 77 77'
 '77 77 80 AA A0 88 77 77 00 00 77 71 77 77 78 0A'
 'A0 88 77 77 00 00 77 19 77 77 77 80 A0 88 77 77'
 '00 00 77 97 77 77 77 78 00 88 79 77 00 00 71 77'
 '77 77 77 77 88 87 71 19 00 00 19 77 77 77 77 77'
 '77 77 77 71 00 00'
}


CM_FILEEXIT BITMAP
{
 '42 4D 66 01 00 00 00 00 00 00 76 00 00 00 28 00'
```

```
 '00 00 14 00 00 00 14 00 00 00 01 00 04 00 00 00'
 '00 00 F0 00 00 00 00 00 00 00 00 00 00 00 00 00'
 '00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 80'
 '00 00 00 80 80 00 80 00 00 00 80 00 80 00 80 80'
 '00 00 C0 C0 C0 00 80 80 80 00 00 00 FF 00 00 FF'
 '00 00 00 FF FF 00 FF 00 00 00 FF 00 FF 00 FF FF'
 '00 00 FF FF FF 00 EE EE EE EE EE EE EE EE EE EE'
 '00 00 EE EE FF EF EF FE EE EE EE EE 00 00 EE EE'
 'FF EE FE EF EE EE EE EE 00 00 EE EE FF EF FF FF'
 'EE EE EE EE 00 00 EE EE EF EE EF FF EE EE EE EE'
 '00 00 88 88 88 88 88 88 88 88 88 88 00 00 66 66'
 '8A AA A8 77 86 66 66 66 00 00 66 66 8A AA A8 77'
 '86 66 66 66 00 00 66 66 8A AA A8 77 86 66 66 66'
 '00 00 66 66 8A A9 A8 77 86 66 66 66 00 00 66 66'
 '8A A9 A8 77 86 66 66 66 00 00 66 66 8A AA A8 77'
 '86 66 66 66 00 00 66 66 8A AA A8 77 86 66 66 66'
 '00 00 66 66 8A AA A8 77 86 66 66 66 00 00 66 66'
 '8A A8 87 77 86 66 66 66 00 00 66 66 8A 87 77 77'
 '86 66 66 66 00 00 66 66 88 77 77 77 86 66 66 66'
 '00 00 66 66 88 88 88 88 86 66 66 66 00 00 66 66'
 '66 66 66 66 66 66 66 66 00 00 66 66 66 66 66 66'
 '66 66 66 66 00 00'
}


CM_ABOUT BITMAP
{
 '42 4D 66 01 00 00 00 00 00 00 76 00 00 00 28 00'
 '00 00 14 00 00 00 14 00 00 00 01 00 04 00 00 00'
 '00 00 F0 00 00 00 00 00 00 00 00 00 00 00 00 00'
 '00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 80'
 '00 00 00 80 80 00 80 00 00 00 80 00 80 00 80 80'
 '00 00 C0 C0 C0 00 80 80 80 00 00 00 FF 00 00 FF'
 '00 00 00 FF FF 00 FF 00 00 00 FF 00 FF 00 FF FF'
 '00 00 FF FF FF 00 BB BB BB BB BB BB BB BB BB BB'
 '00 00 BB BB BB BB BB BB BB BB BB BB 00 00 BB BB'
 'BB BB BB BB BB BB BB BB 00 00 BB BB BB B8 88 88'
 'BB BB BB BB 00 00 BB BB BB B8 88 88 BB BB BB BB'
 '00 00 BB BB BB B8 00 88 BB BB BB BB 00 00 BB BB'
 'BB B8 00 BB BB BB BB BB 00 00 BB BB BB BB BB BB'
 'BB BB BB BB 00 00 BB BB BB B8 00 88 BB BB BB BB'
 '00 00 BB BB BB BB 80 08 BB BB BB BB 00 00 BB BB'
 'B8 8B B8 00 8B BB BB BB 00 00 BB BB B8 00 BB 00'
 '8B BB BB BB 00 00 BB BB B8 00 BB 00 8B BB BB BB'
 '00 00 BB BB BB 80 00 00 8B BB BB BB 00 00 BB BB'
 'BB B8 00 00 8B BB BB BB 00 00 BB BB BB B8 88 88'
 'BB BB BB BB 00 00 BB BB BB BB BB BB BB BB BB BB'
 '00 00 BB BB BB BB BB BB BB BB BB BB 00 00 BB BB'
 'BB BB BB BB BB BB BB BB 00 00 BB BB BB BB BB BB'
 'BB BB BB BB 00 00'
}

CM_INPUTFILESITEM1 BITMAP "pointfile.bmp"
/*{
 '42 4D 96 01 00 00 00 00 00 00 76 00 00 00 28 00'
 '00 00 18 00 00 00 18 00 00 00 01 00 04 00 00 00'
 '00 00 20 01 00 00 00 00 00 00 00 00 00 00 00 00'
 '00 00 10 00 00 00 00 00 00 00 00 00 80 00 00 80'
 '00 00 00 80 80 00 80 00 00 00 80 00 80 00 80 80'
 '00 00 C0 C0 C0 00 80 80 80 00 00 00 FF 00 00 FF'
 '00 00 00 FF FF 00 FF 00 00 00 FF 00 FF 00 FF FF'
```

```
 '00 00 FF FF FF 00 FF FF FF FF FF FF FF FF FF'
 'FF FF FF 00 00 00 00 00 00 00 00 08 8F FF 0A'
 'AA AA AA AA AA AA AA AA 08 8F FF 0A AA AA AA AA'
 'AA AA AA AA 08 8F FF 0A 99 9A AA AA AA AA AA AA'
 '08 8F FF 0A AA AA AA AA AA AA AA AA 08 8F FF 0A'
 'AA AA AA AA AA AA AA AA 08 8F FF 00 00 00 00 00'
 '00 00 00 00 08 8F FF 88 88 88 88 88 88 88 88 88'
 '88 8F FF FF FF FF FF FF FF FF FF FF FF FF FF FF'
 'FF FF FF FF FF FF FF FF FF FF FF F0 00 00 00 00'
 '00 00 0F FF FF FF FF F0 FF FF FF FF FF FF 0F FF'
 'FF FF FF F0 F0 00 FF FF FF FF 0F FF FF FF FF F0'
 'FF FF FF F0 00 0F 0F FF FF FF FF F0 FF FF FF FF'
 'FF FF 0F FF FF FF FF F0 FF FF FF 00 0F FF 0F FF'
 'FF FF FF F0 FF FF FF FF FF F0 8F FF FF FF FF F0'
 'F0 00 FF FF FF 08 8F FF FF FF FF F0 FF FF FF FF'
 '00 88 8F FF FF FF FF F0 FF FF FF F0 88 88 8F FF'
 'FF FF FF F0 00 00 00 08 88 88 8F FF FF FF FF FF'
 'FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF'
 'FF FF FF FF FF FF'
}
*/

CM_POPUPITEM1 BITMAP "tinfile.bmp"
/*{
 '42 4D 96 01 00 00 00 00 00 00 76 00 00 00 28 00'
 '00 00 18 00 00 00 18 00 00 00 01 00 04 00 00 00'
 '00 00 20 01 00 00 00 00 00 00 00 00 00 00 00 00'
 '00 00 10 00 00 00 00 00 00 00 00 00 80 00 00 80'
 '00 00 00 80 80 00 80 00 00 00 80 00 80 00 80 80'
 '00 00 C0 C0 C0 00 80 80 80 00 00 00 FF 00 00 FF'
 '00 00 00 FF FF 00 FF 00 00 00 FF 00 FF 00 FF FF'
 '00 00 FF FF FF 00 FF FF FF FF FF FF FF FF FF FF'
 'FF FF FF 00 00 00 00 00 00 00 00 00 08 8F FF 0A'
 'AA AA AA AA AA AA AA AA 08 8F FF 0A AA AA AA AA'
 'AA AA AA AA 08 8F FF 0A A9 99 9A AA AA AA AA AA'
 '08 8F FF 0A AA AA AA AA AA AA AA AA 08 8F FF 0A'
 'AA AA AA AA AA AA AA AA 08 8F FF 00 00 00 00 00'
 '00 00 00 00 08 8F FF 88 88 88 88 88 88 88 88 88'
 '88 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF'
 'FF FF FF FF FF FF FF FF FF FF FF FF 00 00 00 00'
 '00 00 00 FF FF FF FF FF 0F FF FF FF FF FF F0 FF'
 'FF FF FF FF 0F FF FF FF 00 FF F0 FF FF FF FF FF'
 '0F FF FF FF FF FF F0 FF FF FF FF FF 0F F0 00 0F'
 'FF FF F0 FF FF FF FF FF 0F FF FF FF FF FF F0 FF'
 'FF FF FF FF 0F FF FF F0 0F FF F0 FF FF FF FF FF'
 '0F FF FF FF FF FF 08 FF FF FF FF FF 0F F0 00 FF'
 'FF F0 88 FF FF FF FF FF 0F FF FF FF F0 08 88 FF'
 'FF FF FF FF 0F FF FF FF 08 88 88 FF FF FF FF FF'
 '00 00 00 00 88 88 88 FF FF FF FF FF FF FF FF FF'
 'FF FF FF FF FF FF'
}

*/
```

3D Rasterization Code

The following gives the complete code for the 3D rasterization program.

```
//*********************************************************************//
// A program to rasterise points in 3D                                 //
//      Input: XYZ coordinate                                          //
//      Output: 3D array ILWIS look-like .MPD and .MPI file format     //
//                                                                     //
// ********************************************************************//

#include <fstream.h>
#include <stdlib.h>
#include <stdio.h>
#include <iomanip.h>
#include <io.h>
#include <conio.h>
#include <string>


const int rowsize = 190;
const int colsize = 190;
const int zsize   = 190;

typedef struct PointStruct           // structure for XYZ world coordinates
  {
   float x;
   float y;
   float z;
  } PointType;

PointType Point;

typedef struct VpiStruct
  {
   short Nscanlines;      // no. of image rows
   short Npixels;         // no. of image cols
   short Nlevels;         // no. of image levels
   short Pvmin;           // voxel's minimum value
   short Pvmax;           // voxel's maximum value
   short Maptype;         // map type
   short Patch;           // patch type
   short Scale;           // image scale factor
   short Crdtype;         // coordinate type
   float a11;             // transformation cooefficient (3 x 3 matrix)
   float a12;             //                  "
   float a13;             //                  "
   float a21;             //                  "
   float a22;             //                  "
   float a23;             //                  "
   float a31;             //                  "
```

```
   float a32;             //                 "
   float a33;             //                 "
   float b1;              // translation vector (3 x 1 matrix)
   float b2;              //                 "
   float b3;              //                 "
  } VpiType;

VpiType VPI;

// typedef section
typedef short Image[rowsize][colsize][zsize];    // Image: Voxel variable

Image Voxel;

float xlength, ylength;
float zlength;
int voxelsize;
short maxrow, maxcol, maxlevel;
short npnt, pnt, maxpnt;
float scalex, scaley, scalez, scale;
float xmin, ymin, xmax, ymax;
float zmin, zmax;
FILE* VPDfile;
FILE* VPIfile;


// prototype section
void GetXYZfile(float&, float&, float&, float&, float&, float&);
void WriteVpiFile();
void GetVoxelSize(int&);
void GetRowColLevel(int, float&, float&, float&);
void Make3DRaster(PointType, int, float, float, float);
void DisplayVoxel();
void PressAnyKey();

// Main
void main()
  {
   GetXYZfile(xmin, ymin, zmin, xmax, ymax, zmax);
   GetVoxelSize(voxelsize);
   GetRowColLevel(voxelsize, xlength, ylength, zlength);
   Make3DRaster(Point, voxelsize, xlength, ylength, zlength);
   WriteVpiFile();
   fclose(VPDfile);
   fclose(VPIfile);
   PressAnyKey();
  }

// definition section
void GetXYZfile(float& xmin, float& ymin, float& zmin,
                float& xmax, float& ymax, float& zmax)
  {
   ifstream XYZfile("c:\\data\\lakebore.xyz");

     if (! XYZfile)
       {
        cerr << "Point file not found" << endl;
        exit(1);
       }
```

```
   npnt = 0;

   // reads the file header
   string charX, charY, charZ;
   XYZfile >> charX >> charY >> charZ;
   XYZfile >> Point.x >> Point.y >> Point.z;

   // to calculate min xyz, max xyz
   xmin = Point.x;
   ymin = Point.y;
   zmin = Point.z;

   xmax = Point.x;
   ymax = Point.y;
   zmax = Point.z;

   do
     {
      npnt ++;
      printf("\r");
      cout << "Reading points ... " << npnt;

      XYZfile >> Point.x >> Point.y >> Point.z;
        if (xmin > Point.x)
            xmin = Point.x;

        if (ymin > Point.y)
            ymin = Point.y;

        if (zmin > Point.z)
            zmin = Point.z;

        if (xmax < Point.x)
            xmax = Point.x;

        if (ymax < Point.y)
            ymax = Point.y;

        if (zmax < Point.z)
            zmax = Point.z;

     } while (! XYZfile.eof());
   npnt = npnt - 1;

   xlength = xmax - xmin;
   ylength = ymax - ymin;
   zlength = zmax - zmin;

   XYZfile.close();
  }

void GetVoxelSize(int& voxelsize)
  {
   cout << endl;
   cout << "Enter voxel size (in meter): " << endl;
   cin >> voxelsize;
  }

void GetRowColLevel(int voxelsize,
                    float& xlength, float& ylength, float& zlength)
  {
```

```
   xlength = xmax - xmin;
   ylength = ymax - ymin;
   zlength = zmax - zmin;

   maxrow   = int (ylength / voxelsize) + 1;
   maxcol   = int (xlength / voxelsize) + 1;
   maxlevel = int (zlength / voxelsize) + 1;
  }

void Make3DRaster(PointType Point, int voxelsize,
                  float xlength, float ylength, float zlength)
  {
   short PointID;
   int row, col, level;

   scale = 1.0 / float(voxelsize);

   cout << endl;
   cout << "xmin, ymin, zmin: " << xmin << ", " << ymin << ", " << zmin <<
endl;
   cout << "xmax, ymax, zmax: " << xmax << ", " << ymax << ", " << zmax <<
endl;
   cout << "voxel size : " << voxelsize << endl;
   cout << "xlength : " << xlength << endl;
   cout << "ylength : " << ylength << endl;
   cout << "zlength : " << zlength << endl;
   cout << "maxrow  :" << maxrow << endl;
   cout << "maxcol  :" << maxcol << endl;
   cout << "maxzrow :" << maxlevel << endl;

   cout << setiosflags(ios::showpoint) << endl;
   cout << setprecision(5) << "scale   : " << scale << endl;

   ifstream XYZfile("c:\\data\\lakebore.xyz");

   if (! XYZfile)
       {
        cerr << "Point file not found" << endl;
        exit(1);
       }

   string charX, charY, charZ;
   FILE* VPDfile;
   char vpdfilename[] = "c:\\data\\lakebo.vpd";
   VPDfile = fopen(vpdfilename, "wb");

   pnt = -1;

   // reads the file header
   XYZfile >> charX >> charY >> charZ;
   while (! XYZfile.eof())
     {
      pnt ++;
      printf("\r");
      cout << "Rasterizing the points ... " << pnt;
      XYZfile >> Point.x >> Point.y >> Point.z;
      col  = int (scale * (Point.x - xmin)) + 1;
      row  = int (scale * (Point.y - ymin)) + 1;
      //level = int ((-1 * scale) * (Point.z - zmin)) + maxlevel + 1;
      level = int (scale * (Point.z - zmin)) + 1;
      PointID = short(pnt-1);
```

```
      Voxel[level][row][col] = PointID;
     }

/*   do
     {
      printf("\r");
      cout << "Rasterizing the points ... " << pnt;
      XYZfile >> Point.x >> Point.y >> Point.z;
      PointID = short(pnt);
      col  = int (scale * (Point.x - xmin)) + 1;
      row  = int (scale * (Point.y - ymin)) + 1;
      level = int ((-1 * scale) * (Point.z - zmin)) + maxlevel + 1;
      Voxel[level][row][col] = PointID;
      pnt ++;
     } while (! XYZfile.eof());
*/
   for (level = 0; level < maxlevel; level ++)
     {
      for (row = 0; row < maxrow; row ++)
        {
         for (col = 0; col < maxcol; col ++)
           {
            fwrite(&Voxel[level][row][col], sizeof(short), 1, VPDfile);
           }
        }
     }

   cout << endl;
   cout << endl;
   cout << "Total points : " << npnt;
   cout << endl;
   cout << endl;
   XYZfile.close();
  }

void DisplayVoxel()
  {
   int l, r, c;

   cout << "Displaying voxel value ..." << endl;
   cout << endl;

   for (l = 0; l < maxlevel; l ++)
     {
      for (r = 0; r < maxrow; r ++)
        {
         for (c = 0; c < maxcol; c ++)
           {
            printf("\r");
            cout << "Voxel[zrow][row][col] : Voxel[" << l << "][" << r <<
                 "][" << c << "] : " << Voxel[l][r][c];
           }
        }
     }


  }

void WriteVpiFile()
  {
   FILE* VPIfile;
```

```
   char vpifilename[] = "c:\\data\\lakebo.vpi";
   VPIfile = fopen(vpifilename, "wb");

   // assign the value for the VPI file
   VPI.Nscanlines = maxrow;
   VPI.Npixels = maxcol;
   VPI.Nlevels = maxlevel;
   VPI.Pvmin = 0;
   VPI.Pvmax = npnt;
   VPI.Maptype = 2;
   VPI.Patch = 0;
   VPI.Scale = 0;
   VPI.Crdtype = 1;
   VPI.a11 = scale;
   VPI.a12 = 0;
   VPI.a13 = 0;
   VPI.a21 = 0;
   VPI.a22 = scale;
   VPI.a23 = 0;
   VPI.a31 = 0;
   VPI.a32 = 0;
   VPI.a33 = -scale;
   VPI.b1 = 1;
   VPI.b2 = 1;
   VPI.b3 = maxlevel + 1;

   // write out the above values to the VPI file
   fwrite(&VPI, sizeof(VpiType), 1, VPIfile);
  }

void PressAnyKey()
  {
   cout << endl;
   cout << endl;
   cout << "Press any key to continue" << endl;
   getch();
   cout << endl;
  }
```

3D Distance Transformation Code

The following gives the complete codes for the 3D Distance Transformation program.

```
//*************************************************************************
**//
//     Object-Oriented 3D Distance Transformation (DT) program for 3D TIN
//
//               Input  :  3D array format
//
//               Output :  3D array ILWIS look-like file format
//
//
//
//*************************************************************************
**//

#include <iostream.h>
#include <stdlib.h>
#include <stdio.h>
#include <fstream.h>
#include <io.h>
#include <iomanip.h>
#include <dos.h>
#include <malloc.h>
#include <conio.h>
#include <limits.h>
#include <stddef.h>

// Constant section

#define masksize 27           // Mask size

class DistanceTransform3D
  {
   public:
        DistanceTransform3D();
        ~DistanceTransform3D();

       // Data members
       // VPI input file structure
       typedef struct VpiStruct
                 {
                  short Nscanlines;      // no. of image rows
                  short Npixels;         // no. of image cols
                  short Nlevels;         // no. of image levels
                  short Pvmin;           // voxel's minimum value
                  short Pvmax;           // voxel's maximum value
                  short Maptype;         // map type
                  short Patch;           // patch type
                  short Scale;           // image scale factor
```

```
                short Crdtype;          // coordinate type
                float a11;              // transformation cooefficient (3 x
3 matrix)
                float a12;              //                  "
                float a13;              //                  "
                float a21;              //                  "
                float a22;              //                  "
                float a23;              //                  "
                float a31;              //                  "
                float a32;              //                  "
                float a33;              //                  "
               float b1;               // translation vector (3 x 1 matrix)
                float b2;               //                  "
                float b3;               //                  "
               } VpiType;

      VpiType VPI, VPIOut;

      typedef short Image;
      typedef Image* VoxelRow;
      typedef VoxelRow* Voxel2D;
      typedef Voxel2D* Voxel3D;

      typedef short Mask[masksize];

      // Global variables
      Mask MaskPix;
      FILE* VPDifile;
      FILE* VPDofile;
      FILE* VPIifile;
      FILE* VPIofile;

      Voxel3D Voxel;           // variable for voxels

      int maxrow;
      int maxcol;
      int maxlevel;
      float scale;
      int row, col, level;
      int r, l;

      // Function members
      void GetVPDInputFile();
      void GetVPIInputFile(VpiType& VPI);
      void ReadVoxelImage(Voxel3D&);
      void PrintVoxel(Voxel3D);
      void WriteVPDOutputFile();
      void WriteVPIOutputFile();
      void SetBackground(Voxel3D, int, int);
      void GetUpperMask(int, int, int, Voxel3D, Mask&);
      void GetLowerMask(int, int, int, Voxel3D, Mask&);
      int MinByIndex(int, int);
      int Min5(int, int, int, int, int);
      void DistancePassOne(Voxel3D);
      void DistancePassTwo(Voxel3D);
      void ForwardDistance();
      void BackwardDistance();
      void DisplayVoxel(Voxel3D);
      void PressAnyKey();
```

```
        void Title();
    };

// Main program
void main()
  {
   DistanceTransform3D ThreeDDT;    // ThreeDDT is an object
   ThreeDDT.Title();
   ThreeDDT.GetVPDInputFile();
   ThreeDDT.ForwardDistance();
   ThreeDDT.BackwardDistance();
   ThreeDDT.WriteVPDOutputFile();
   ThreeDDT.WriteVPIOutputFile();
   ThreeDDT.PressAnyKey();
  }

// End of main program

// Definitions sections follows:
// The constructor
DistanceTransform3D :: DistanceTransform3D()
  {
   GetVPIInputFile(VPI);

   // Allocate memory dynamically
   Voxel = new Voxel2D[maxlevel];
   for (l = 0; l < maxlevel; l ++)
     {
      Voxel[l] = new VoxelRow[maxrow];
      for (r = 0; r < maxrow; r ++)
        {
         Voxel[l][r] = new Image[maxcol];
        }
     }
  }

// The destructor
DistanceTransform3D :: ~DistanceTransform3D()
  {
   // Deallocate memory
   for (l = 0; l < maxlevel; l ++)
     {
      for (r = 0; r < maxrow; r ++)
        {
         delete [] Voxel[l][r];
        }
     }
   delete [] Voxel;
  }

// Procedure: PressAnyKey
void DistanceTransform3D :: PressAnyKey()
  {
   cout << endl;
   cout << "Press any key to continue" << endl;
   getch();
   cout << endl;
  }


// Procedure: GetVoxelInputFile
```

```
void DistanceTransform3D :: GetVPDInputFile()
  {
   char vpdifilename [] = "c:\\data\\ras3d.vpd";

   VPDifile = fopen(vpdifilename, "rb");
   if (VPDifile == NULL)
     {
      cerr << "Could not open VPD file !" << endl;
      exit(1);
     }
   else
     {
      cout << endl;
      cout << "The VPD file is opened" << endl;
     }
  }

// Procedure: GetVPIInputFile
void DistanceTransform3D :: GetVPIInputFile(VpiType& VPI)
  {
   char vpiifilename[] = "c:\\data\\ras3d.vpi";
   VPIifile = fopen(vpiifilename, "rb");
   if (VPIifile == NULL)
     {
      cerr << "Could not open VPI file !" << endl;
      exit(1);
     }
   else
     {
      cout << endl;
      cout << "Information of  " << vpiifilename << endl;
      fread(&VPI, sizeof(VpiType), 1, VPIifile);
      int size = sizeof(VpiType);

      maxrow   = VPI.Nscanlines;
      maxcol   = VPI.Npixels;
      maxlevel = VPI.Nlevels;

      cout << "VPI file size       : " << size << endl;
      cout << endl;
      cout << "Number of lines     : " << VPI.Nscanlines << endl;
      cout << "Number of columns   : " << VPI.Npixels << endl;
      cout << "Number of levels    : " << VPI.Nlevels << endl;
      cout << "Minimum value       : " << VPI.Pvmin << endl;
      cout << "Maximum value       : " << VPI.Pvmax << endl;
      cout << "Map type            : " << VPI.Maptype << endl;
      cout << "Patched             : " << VPI.Patch << endl;
      cout << "Scale (power of 10) : " << VPI.Scale << endl;
      cout << "Coordinate type     : " << VPI.Crdtype << endl;
      cout << "Transformation parameters :" << endl;
      cout << "   a11  : " << VPI.a11 << endl;
      cout << "   a12  : " << VPI.a12 << endl;
      cout << "   a13  : " << VPI.a31 << endl;
      cout << "   a21  : " << VPI.a21 << endl;
      cout << "   a22  : " << VPI.a22 << endl;
      cout << "   a23  : " << VPI.a23 << endl;
      cout << "   a31  : " << VPI.a31 << endl;
      cout << "   a32  : " << VPI.a32 << endl;
      cout << "   a33  : " << VPI.a33 << endl;
      cout << endl;
      cout << "Translation vectors : " << endl;
```

```
      cout << "   b1   : " << VPI.b1 << endl;
      cout << "   b2   : " << VPI.b2 << endl;
      cout << "   b3   : " << VPI.b3 << endl;
      fclose(VPIifile);
     }
  }

// Procedure: ReadVoxelImage
void DistanceTransform3D :: ReadVoxelImage(Voxel3D& Voxel)
  {
   int r;
   int c;
   int l;

   cout << endl;
   cout << "Reading voxels data ..., wait !" << endl;
   cout << endl;

   for (l = 0; l < maxlevel; l ++)
     {
      for (r = 0; r < maxrow; r ++)
        {
         for (c = 0; c < maxcol; c ++)
           {
            fread(&Voxel[l][r][c], sizeof(short), 1, VPDifile);
           }
        }
     }
   cout << endl;
   fclose(VPDifile);
  }

// Procedure: PrintPixel
void DistanceTransform3D :: PrintVoxel(Voxel3D Voxel)
  {
   int r;
   int c;
   int l;

   for (r = 0; r < maxrow; r ++)
     {
      for (c = 0; c < maxcol; c ++)
        {
         for (l = 0; l < maxlevel; l ++)
           {
            cout << "Voxel value at [row][col][level]: [" << r << "][" << c
                 << "][" << l << "]: " << Voxel[r][c][l] << "\r";
           }
        }
     }
  }


// Procedure: WriteVoxelOutputFile
void DistanceTransform3D :: WriteVPDOutputFile()
  {
   int r;
   int l;
   int c;
```

```
   char vpdofilename[] = "c:\\data\\voxodcc.vpd";
   VPDofile = fopen(vpdofilename, "w+b");
   for (l = 0; l < maxlevel; l ++)
     {
      for (r = 0; r < maxrow; r ++)
        {
         for (c = 0; c < maxcol; c ++)
           {
            fwrite(&Voxel[l][r][c], sizeof(short), 1, VPDofile);
           }
        }
     }
   fclose(VPDofile);
  }

// Procedure: WriteVPIFile
void DistanceTransform3D :: WriteVPIOutputFile()
  {
   FILE* VPIofile;
   char vpifilename[] = "c:\\data\\voxodcc.vpi";
   VPIofile = fopen(vpifilename, "wb");

   // assign the value for the VPI file
   VPIOut.Nscanlines = VPI.Nscanlines;
   VPIOut.Npixels = VPI.Npixels;
   VPIOut.Nlevels = VPI.Nlevels;
   VPIOut.Pvmin = 0;
   VPIOut.Pvmax = 6000;
   VPIOut.Maptype = 2;
   VPIOut.Patch = 0;
   VPIOut.Scale = 0;
   VPIOut.Crdtype = 1;
   VPIOut.a11 = VPI.a11;
   VPIOut.a12 = 0;
   VPIOut.a13 = 0;
   VPIOut.a21 = 0;
   VPIOut.a22 = VPI.a22;
   VPIOut.a23 = 0;
   VPIOut.a31 = 0;
   VPIOut.a32 = 0;
   VPIOut.a33 = VPI.a33;
   VPIOut.b1 = 1;
   VPIOut.b2 = 1;
   VPIOut.b3 = maxlevel + 1;

   // write out the above values to the VPI file
   fwrite(&VPIOut, sizeof(VpiType), 1, VPIofile);
   fclose(VPIofile);
  }


// Procedure: SetBackground
void DistanceTransform3D :: SetBackground(Voxel3D Voxel, int Bg, int Fg)
  {
   int row;
   int col;
   int l;

   for (l = 0; l < maxlevel; l ++)
     {
      for (row = 0; row < maxrow; row ++)
```

```
        {
         for (col = 0; col < maxcol; col ++)
           {
            if (Voxel[l][row][col] == 0)
                Voxel[l][row][col] = Bg;
            else
            if (Fg > 0)
                Voxel[l][row][col] = Fg;
            }

        }
     }
  }

// Procedure: GetUpperMask
void DistanceTransform3D :: GetUpperMask(int l, int r, int c, Voxel3D Voxel,
Mask& MaskPix)
  {
   MaskPix[0]  = Voxel[l-1][r-1][c-1];
   MaskPix[1]  = Voxel[l-1][r-1][c];
   MaskPix[2]  = Voxel[l-1][r-1][c+1];
   MaskPix[3]  = Voxel[l-1][r][c-1];
   MaskPix[4]  = Voxel[l-1][r][c];
   MaskPix[5]  = Voxel[l-1][r][c+1];
   MaskPix[6]  = Voxel[l-1][r+1][c-1];
   MaskPix[7]  = Voxel[l-1][r+1][c];
   MaskPix[8]  = Voxel[l-1][r+1][c+1];
   MaskPix[9]  = Voxel[l][r-1][c-1];
   MaskPix[10] = Voxel[l][r-1][c];
   MaskPix[11] = Voxel[l][r-1][c+1];
   MaskPix[12] = Voxel[l][r][c-1];
   MaskPix[13] = Voxel[l][r][c];

  }

// Procedure: GetLowerMask
void DistanceTransform3D :: GetLowerMask(int l, int r, int c, Voxel3D Voxel,
Mask& MaskPix)
  {
   MaskPix[13] = Voxel[l][r][c];
   MaskPix[14] = Voxel[l][r][c+1];
   MaskPix[15] = Voxel[l][r+1][c-1];
   MaskPix[16] = Voxel[l][r+1][c];
   MaskPix[17] = Voxel[l][r+1][c+1];
   MaskPix[18] = Voxel[l+1][r-1][c-1];
   MaskPix[19] = Voxel[l+1][r-1][c];
   MaskPix[20] = Voxel[l+1][r+1][c+1];
   MaskPix[21] = Voxel[l+1][r][c-1];
   MaskPix[22] = Voxel[l+1][r][c];
   MaskPix[23] = Voxel[l+1][r][c+1];
   MaskPix[24] = Voxel[l+1][r+1][c-1];
   MaskPix[25] = Voxel[l+1][r+1][c];
   MaskPix[26] = Voxel[l+1][r+1][c+1];

  }

// Function: MinByIndex
int DistanceTransform3D :: MinByIndex(int from, int to)
  {
   int i;
   int temp;
```

```
   int Idx;

   temp = MaskPix[from];
   Idx = from;
   for (i = from; i <= to ; i ++)
     {
      if (temp > MaskPix[i])
        {
         temp = MaskPix[i];
         Idx = i;
        }
     }
   return Idx;
  }

#define min(a, b) (((a) < (b)) ? (a) : (b))

// Function Minimum of 5 values
int DistanceTransform3D :: Min5(int a, int b, int c, int d, int e)
  {
   int mn;

   mn = min(a, b);
   mn = min(mn, c);
   mn = min(mn, d);
   mn = min(mn, e);
   return mn;
  }

// Procedure: DistancePassOne
void DistanceTransform3D :: DistancePassOne(Voxel3D Voxel)
  {
   int r;
   int c;
   int l;
   int k;

   for (l = 1; l < maxlevel-1; l ++)
     {
      for (r = 1; r < maxrow-1; r ++)
        {
         for (c = 1; c < maxcol-1; c ++)
           {
            GetUpperMask(l, r, c, Voxel, MaskPix);
            for (k = 0; k < 13; k ++)
               {
                if ((k == 0) || (k == 2) ||
                    (k == 6) || (k == 8))
                   MaskPix[k] = MaskPix[k] + 5;

                if ((k == 1) || (k == 3) ||
                    (k == 5) || (k == 7) ||
                    (k == 9) || (k == 11))
                   MaskPix[k] = MaskPix[k] + 4;

                if ((k == 4) || (k == 10) || (k == 12))
                   MaskPix[k] = MaskPix[k] + 3;
                }

            if (MaskPix[13] != 30000)
```

```
                MaskPix[13] = 0;

             Voxel[l][r][c] = MaskPix[MinByIndex(0, 13)];
             //printf("\r");
             //cout << "level: " << l << " row: " << r << " col: " << c;
             //cout << "  Voxel[l][r][c]: " << Voxel[l][r][c];
            }
         }
      }
  }

// Procedure: DistancePassTwo
void DistanceTransform3D :: DistancePassTwo(Voxel3D Voxel)
  {
   int r;
   int c;
   int l;
   int k;

   for (l = maxlevel - 2; l > 0; l --)
     {
      for (r = maxrow - 2; r > 0; r --)
        {
         for (c = maxcol - 2; c > 0; c --)
            {
             GetLowerMask(l, r, c, Voxel, MaskPix);
             for (k = 26; k > 13; k --)
               {
                 if ((k == 18) || (k == 20) ||
                     (k == 24) || (k == 26))
                    MaskPix[k] = MaskPix[k] + 5;

                 if ((k == 19) || (k == 21) ||
                     (k == 23) || (k == 25) ||
                     (k == 15) || (k == 17))
                    MaskPix[k] = MaskPix[k] + 4;

                 if ((k == 14) || (k == 16) || (k == 22))
                    MaskPix[k] = MaskPix[k] + 3;
               }
             Voxel[l][r][c] = MaskPix[MinByIndex(13, 26)];
             }
         }
      }
   }


// Procedure: ForwardDistance
void DistanceTransform3D :: ForwardDistance()
  {
   cout << "3D Forward Distance ..." << endl;
   ReadVoxelImage(Voxel);
   SetBackground(Voxel, 30000, 0);
   DistancePassOne(Voxel);
  }

// Procedure: BackwardDistance
void DistanceTransform3D :: BackwardDistance()
  {
   cout << "3D Backward Distance ..." << endl;
   DistancePassTwo(Voxel);
```

```
  }


// Procedure: Title
void DistanceTransform3D :: Title()
  {
   cout << endl;
      cout  <<  "---------------Object-Oriented  3D  Distance  Transformation
--------------" << endl;
   cout << "----------- By: Alias Abdul-Rahman, November, 1998 ------------"
<< endl;
   cout << "   Input/Output : 3D array ILWIS look-like file format. " << endl;
   cout << endl;
  }
```

```
// Object-Oriented 3D DistanceTransform Program
// File: TDistanceTransform3D.h

#include "TDistanceTransform3D.h"

// Main program
void main()
  {
   TDistanceTransform3D ThreeDDT;    // ThreeDDT is an object
   ThreeDDT.Title();
   ThreeDDT.GetVPDInputFile();
   ThreeDDT.ForwardDistance();
   ThreeDDT.BackwardDistance();
   ThreeDDT.WriteVPDOutputFile();
   ThreeDDT.WriteVPIOutputFile();
   ThreeDDT.PressAnyKey();
  }

// End of main program

// Definitions sections follows:
// The constructor
TDistanceTransform3D :: TDistanceTransform3D()
  {
   GetVPIInputFile(VPI);

   // Allocate memory dynamically
   Voxel = new Voxel2D[maxlevel];
   for (l = 0; l < maxlevel; l ++)
     {
      Voxel[l] = new VoxelRow[maxrow];
      for (r = 0; r < maxrow; r ++)
        {
         Voxel[l][r] = new Image[maxcol];
        }
     }
  }

// The destructor
TDistanceTransform3D :: ~TDistanceTransform3D()
  {
   // Deallocate memory
   for (l = 0; l < maxlevel; l ++)
     {
      for (r = 0; r < maxrow; r ++)
        {
         delete [] Voxel[l][r];
        }
     }
   delete [] Voxel;
  }

// Procedure: PressAnyKey
void TDistanceTransform3D :: PressAnyKey()
  {
   cout << endl;
   cout << "Press any key to continue" << endl;
   getch();
```

```
   cout << endl;
  }


// Procedure: GetVoxelInputFile
void TDistanceTransform3D :: GetVPDInputFile()
  {
   char vpdifilename [] = "c:\\data\\pnt5.vpd";

   VPDifile = fopen(vpdifilename, "rb");
   if (VPDifile == NULL)
     {
      cerr << "Could not open VPD file !" << endl;
      exit(1);
     }
   else
     {
      cout << endl;
      cout << "The VPD file is opened" << endl;
     }
  }

// Procedure: GetVPIInputFile
void TDistanceTransform3D :: GetVPIInputFile(VpiType& VPI)
  {
   char vpiifilename[] = "c:\\data\\pnt5.vpi";
   VPIifile = fopen(vpiifilename, "rb");
   if (VPIifile == NULL)
     {
      cerr << "Could not open VPI file !" << endl;
      exit(1);
     }
   else
     {
      cout << endl;
      cout << "Information of  " << vpiifilename << endl;
      fread(&VPI, sizeof(VpiType), 1, VPIifile);
      int size = sizeof(VpiType);

      maxrow   = VPI.Nscanlines;
      maxcol   = VPI.Npixels;
      maxlevel = VPI.Nlevels;

      cout << "VPI file size       : " << size << endl;
      cout << endl;
      cout << "Number of lines     : " << VPI.Nscanlines << endl;
      cout << "Number of columns   : " << VPI.Npixels << endl;
      cout << "Number of levels    : " << VPI.Nlevels << endl;
      cout << "Minimum value       : " << VPI.Pvmin << endl;
      cout << "Maximum value       : " << VPI.Pvmax << endl;
      cout << "Map type            : " << VPI.Maptype << endl;
      cout << "Patched             : " << VPI.Patch << endl;
      cout << "Scale (power of 10) : " << VPI.Scale << endl;
      cout << "Coordinate type     : " << VPI.Crdtype << endl;
      cout << "Transformation parameters :" << endl;
      cout << "   a11  : " << VPI.a11 << endl;
      cout << "   a12  : " << VPI.a12 << endl;
      cout << "   a13  : " << VPI.a31 << endl;
      cout << "   a21  : " << VPI.a21 << endl;
      cout << "   a22  : " << VPI.a22 << endl;
      cout << "   a23  : " << VPI.a23 << endl;
```

```
      cout << "   a31  : " << VPI.a31 << endl;
      cout << "   a32  : " << VPI.a32 << endl;
      cout << "   a33  : " << VPI.a33 << endl;
      cout << endl;
      cout << "Translation vectors : " << endl;
      cout << "   b1   : " << VPI.b1 << endl;
      cout << "   b2   : " << VPI.b2 << endl;
      cout << "   b3   : " << VPI.b3 << endl;
      fclose(VPIifile);
     }
  }

// Procedure: ReadVoxelImage
void TDistanceTransform3D :: ReadVoxelImage(Voxel3D& Voxel)
  {
   int r;
   int c;
   int l;

   cout << endl;
   cout << "Reading voxels data ..., wait !" << endl;
   cout << endl;

   for (l = 0; l < maxlevel; l ++)
     {
      for (r = 0; r < maxrow; r ++)
        {
         for (c = 0; c < maxcol; c ++)
           {
            fread(&Voxel[l][r][c], sizeof(short), 1, VPDifile);
           }
        }
     }
   cout << endl;
   fclose(VPDifile);
  }

// Procedure: PrintPixel
void TDistanceTransform3D :: PrintVoxel(Voxel3D Voxel)
  {
   int r;
   int c;
   int l;

   for (r = 0; r < maxrow; r ++)
     {
      for (c = 0; c < maxcol; c ++)
        {
         for (l = 0; l < maxlevel; l ++)
           {
            cout << "Voxel value at [row][col][level]: [" << r << "][" << c
                 << "][" << l << "]: " << Voxel[r][c][l] << "\r";
           }
        }
     }
  }


// Procedure: WriteVoxelOutputFile
void TDistanceTransform3D :: WriteVPDOutputFile()
  {
```

```
   int r;
   int l;
   int c;

   char vpdofilename[] = "c:\\data\\pnt5od.vpd";
   VPDofile = fopen(vpdofilename, "w+b");
   for (l = 0; l < maxlevel; l ++)
     {
      for (r = 0; r < maxrow; r ++)
        {
         for (c = 0; c < maxcol; c ++)
           {
            fwrite(&Voxel[l][r][c], sizeof(short), 1, VPDofile);
           }
        }
     }
   fclose(VPDofile);
  }

// Procedure: WriteVPIFile
void TDistanceTransform3D :: WriteVPIOutputFile()
  {
   FILE* VPIofile;
   char vpifilename[] = "c:\\data\\pnt5od.vpi";
   VPIofile = fopen(vpifilename, "wb");

   // assign the value for the VPI file
   VPIOut.Nscanlines = VPI.Nscanlines;
   VPIOut.Npixels = VPI.Npixels;
   VPIOut.Nlevels = VPI.Nlevels;
   VPIOut.Pvmin = 0;
   VPIOut.Pvmax = 6000;
   VPIOut.Maptype = 2;
   VPIOut.Patch = 0;
   VPIOut.Scale = 0;
   VPIOut.Crdtype = 1;
   VPIOut.a11 = VPI.a11;
   VPIOut.a12 = 0;
   VPIOut.a13 = 0;
   VPIOut.a21 = 0;
   VPIOut.a22 = VPI.a22;
   VPIOut.a23 = 0;
   VPIOut.a31 = 0;
   VPIOut.a32 = 0;
   VPIOut.a33 = VPI.a33;
   VPIOut.b1 = 1;
   VPIOut.b2 = 1;
   VPIOut.b3 = maxlevel + 1;

   // write out the above values to the VPI file
   fwrite(&VPIOut, sizeof(VpiType), 1, VPIofile);
   fclose(VPIofile);
  }


// Procedure: SetBackground
void TDistanceTransform3D :: SetBackground(Voxel3D Voxel, int Bg, int Fg)
  {
   int row;
   int col;
   int l;
```

```
   for (l = 0; l < maxlevel; l ++)
     {
      for (row = 0; row < maxrow; row ++)
        {
         for (col = 0; col < maxcol; col ++)
           {
            if (Voxel[l][row][col] == 0)
                Voxel[l][row][col] = Bg;
            else
            if (Fg > 0)
                Voxel[l][row][col] = Fg;
            }

        }
     }
  }

// Procedure: GetUpperMask
void TDistanceTransform3D :: GetUpperMask(int l, int r, int c, Voxel3D Voxel,
Mask& MaskPix)
  {
   MaskPix[0]  = Voxel[l-1][r-1][c-1];
   MaskPix[1]  = Voxel[l-1][r-1][c];
   MaskPix[2]  = Voxel[l-1][r-1][c+1];
   MaskPix[3]  = Voxel[l-1][r][c-1];
   MaskPix[4]  = Voxel[l-1][r][c];
   MaskPix[5]  = Voxel[l-1][r][c+1];
   MaskPix[6]  = Voxel[l-1][r+1][c-1];
   MaskPix[7]  = Voxel[l-1][r+1][c];
   MaskPix[8]  = Voxel[l-1][r+1][c+1];
   MaskPix[9]  = Voxel[l][r-1][c-1];
   MaskPix[10] = Voxel[l][r-1][c];
   MaskPix[11] = Voxel[l][r-1][c+1];
   MaskPix[12] = Voxel[l][r][c-1];
   MaskPix[13] = Voxel[l][r][c];

  }

// Procedure: GetLowerMask
void TDistanceTransform3D :: GetLowerMask(int l, int r, int c, Voxel3D Voxel,
Mask& MaskPix)
  {
   MaskPix[13] = Voxel[l][r][c];
   MaskPix[14] = Voxel[l][r][c+1];
   MaskPix[15] = Voxel[l][r+1][c-1];
   MaskPix[16] = Voxel[l][r+1][c];
   MaskPix[17] = Voxel[l][r+1][c+1];
   MaskPix[18] = Voxel[l+1][r-1][c-1];
   MaskPix[19] = Voxel[l+1][r-1][c];
   MaskPix[20] = Voxel[l+1][r+1][c+1];
   MaskPix[21] = Voxel[l+1][r][c-1];
   MaskPix[22] = Voxel[l+1][r][c];
   MaskPix[23] = Voxel[l+1][r][c+1];
   MaskPix[24] = Voxel[l+1][r+1][c-1];
   MaskPix[25] = Voxel[l+1][r+1][c];
   MaskPix[26] = Voxel[l+1][r+1][c+1];

  }

// Function: MinByIndex
int TDistanceTransform3D :: MinByIndex(int from, int to)
```

```
  {
   int i;
   int temp;
   int Idx;

   temp = MaskPix[from];
   Idx = from;
   for (i = from; i <= to ; i ++)
     {
      if (temp > MaskPix[i])
        {
         temp = MaskPix[i];
         Idx = i;
        }
     }
   return Idx;
  }

#define min(a, b) (((a) < (b)) ? (a) : (b))

// Function Minimum of 5 values
int TDistanceTransform3D :: Min5(int a, int b, int c, int d, int e)
  {
   int mn;

   mn = min(a, b);
   mn = min(mn, c);
   mn = min(mn, d);
   mn = min(mn, e);
   return mn;
  }

// Procedure: DistancePassOne
void TDistanceTransform3D :: DistancePassOne(Voxel3D Voxel)
  {
   int r;
   int c;
   int l;
   int k;

   for (l = 1; l < maxlevel-1; l ++)
     {
      for (r = 1; r < maxrow-1; r ++)
        {
         for (c = 1; c < maxcol-1; c ++)
           {
            GetUpperMask(l, r, c, Voxel, MaskPix);
            for (k = 0; k < 13; k ++)
               {
                if ((k == 0) || (k == 2) ||
                    (k == 6) || (k == 8))
                   MaskPix[k] = MaskPix[k] + 5;

                if ((k == 1) || (k == 3) ||
                    (k == 5) || (k == 7) ||
                    (k == 9) || (k == 11))
                   MaskPix[k] = MaskPix[k] + 4;

                if ((k == 4) || (k == 10) || (k == 12))
                   MaskPix[k] = MaskPix[k] + 3;
```

```
                }

            if (MaskPix[13] != 30000)
                MaskPix[13] = 0;

            Voxel[l][r][c] = MaskPix[MinByIndex(0, 13)];
            //printf("\r");
            //cout << "level: " << l << " row: " << r << " col: " << c;
            //cout << "  Voxel[l][r][c]: " << Voxel[l][r][c];
           }
        }
     }
  }

// Procedure: DistancePassTwo
void TDistanceTransform3D :: DistancePassTwo(Voxel3D Voxel)
  {
   int r;
   int c;
   int l;
   int k;

   for (l = maxlevel - 2; l > 0; l --)
     {
      for (r = maxrow - 2; r > 0; r --)
        {
         for (c = maxcol - 2; c > 0; c --)
            {
             GetLowerMask(l, r, c, Voxel, MaskPix);
             for (k = 26; k > 13; k --)
               {
                 if ((k == 18) || (k == 20) ||
                     (k == 24) || (k == 26))
                    MaskPix[k] = MaskPix[k] + 5;

                 if ((k == 19) || (k == 21) ||
                     (k == 23) || (k == 25) ||
                     (k == 15) || (k == 17))
                    MaskPix[k] = MaskPix[k] + 4;

                 if ((k == 14) || (k == 16) || (k == 22))
                    MaskPix[k] = MaskPix[k] + 3;
               }
             Voxel[l][r][c] = MaskPix[MinByIndex(13, 26)];
             }
        }
     }
   }


// Procedure: ForwardDistance
void TDistanceTransform3D :: ForwardDistance()
  {
   cout << "3D Forward Distance ..." << endl;
   ReadVoxelImage(Voxel);
   SetBackground(Voxel, 30000, 0);
   DistancePassOne(Voxel);
  }

// Procedure: BackwardDistance
void TDistanceTransform3D :: BackwardDistance()
```

```
  {
   cout << "3D Backward Distance ..." << endl;
   DistancePassTwo(Voxel);
  }


// Procedure: Title
void TDistanceTransform3D :: Title()
  {
   cout << endl;
      cout  <<  "---------------Object-Oriented  3D  Distance  Transformation
--------------" << endl;
   cout << "----------- By: Alias Abdul-Rahman, November, 1998 ------------"
<< endl;
   cout << "   Input/Output : 3D array ILWIS look-like file format. " << endl;
   cout << endl;
  }
```

3D Voronoi Tessellations Code

The following gives the complete codes for the 3D Voronoi tessellations program.

```
//*********************************************************************//
//          Object-Oriented 3D Distance Transformation (DT) and
//
//                Voronoi Tessellation program for 3D TIN
//
//
//
//              Input  :  3D array format
//
//              Output :  3D array ILWIS look-like file format
//
//                                (i.e. .VPD, and .VPI)
//
//
//
//*********************************************************************//

#include <iostream.h>
#include <stdlib.h>
#include <stdio.h>
#include <fstream.h>
#include <io.h>
#include <iomanip.h>
#include <dos.h>
#include <malloc.h>
#include <conio.h>
#include <limits.h>
#include <stddef.h>

#include "TDistanceTransform3D.h"

// Constant section

#define masksize 27                // Mask size

class TVoronoi3D : public TDistanceTransform3D
  {
   public:
      TVoronoi3D();           // constructor
      ~TVoronoi3D();          // destructor

     typedef short Mask[masksize];

     // Global variables
     Mask MaskPixVor;

     FILE* VPDifile;
     FILE* VPDodfile;
```

```
    FILE* VPDovfile;
    FILE* VPIifile;
    FILE* VPIodfile;
    FILE* VPIovfile;

    // variable definition
    Voxel3D Voxel, VoxelVor;           // variable for voxels

    // Functions Prototype;
    void GetVPDInputFile();
    void GetVPIInputFile(VpiType& VPI);
    void ReadVoxelImage(Voxel3D&);
    void CopyVoxel(Voxel3D, Voxel3D&);
    void PrintVoxel(Voxel3D);
    void WriteVPDOutputFileDist();
    void WriteVPIOutputFileDist();
    void WriteVPDOutputFileVoronoi();
    void WriteVPIOutputFileVoronoi();
    void SetBackground(Voxel3D, int, int);
    void GetUpperMaskDist(int, int, int, Voxel3D, Mask&);
    void GetLowerMaskDist(int, int, int, Voxel3D, Mask&);
    void GetUpperMaskVoronoi(int, int, int, Voxel3D, Mask&);
    void GetLowerMaskVoronoi(int, int, int, Voxel3D, Mask&);
    int MinByIndex(int, int);
    int Min5(int, int, int, int, int);
    void ForwardPass(Voxel3D, Voxel3D);
    void BackwardPass(Voxel3D, Voxel3D);
    void ForwardVoronoi();
    void BackwardVoronoi();
    void PressAnyKey();
    void Title();
};
```

```
// Object-Oriented 3D Voronoi Tessellation Program
// File: TVor3D.cpp

#include "TVor3D.h"

// Main program
void main()
  {
   TVoronoi3D Vor3D;
   Vor3D.Title();
   TVoronoi3D();            // constructor
   Vor3D.GetVPDInputFile();
   Vor3D.ForwardVoronoi();
   Vor3D.BackwardVoronoi();
   Vor3D.WriteVPDOutputFileDist();
   Vor3D.WriteVPIOutputFileDist();
   Vor3D.WriteVPDOutputFileVoronoi();
   Vor3D.WriteVPIOutputFileVoronoi();
   Vor3D.PressAnyKey();
  }

// End of main program

// Definitions sections
// The constructor
TVoronoi3D :: TVoronoi3D()
  {
   GetVPIInputFile(VPI);

   // Allocate memory dynamically for DT
   Voxel = new Voxel2D[maxlevel];
   for (l = 0; l < maxlevel; l ++)
     {
      Voxel[l] = new VoxelRow[maxrow];
      for (r = 0; r < maxrow; r ++)
        {
         Voxel[l][r] = new Image[maxcol];
        }
     }

   // Allocate memory dynamically for Voronoi
   VoxelVor = new Voxel2D[maxlevel];
   for (l = 0; l < maxlevel; l ++)
     {
      VoxelVor[l] = new VoxelRow[maxrow];
      for (r = 0; r < maxrow; r ++)
        {
         VoxelVor[l][r] = new Image[maxcol];
        }
     }
  }

// The destructor
TVoronoi3D :: ~TVoronoi3D()
  {
   // Deallocate memory for DT
   for (l = 0; l < maxlevel; l ++)
     {
      for (r = 0; r < maxrow; r ++)
        {
         delete [] Voxel[l][r];
```

```
        }
      }
    delete [] Voxel;

    // Deallocate memory for Voronoi
    for (l = 0; l < maxlevel; l ++)
      {
       for (r = 0; r < maxrow; r ++)
         {
          delete [] VoxelVor[l][r];
         }
      }
    delete [] VoxelVor;
   }

// Procedure: PressAnyKey
void TVoronoi3D :: PressAnyKey()
   {
    cout << endl;
    cout << "Press any key to continue" << endl;
    getch();
    cout << endl;
   }


// Procedure: GetVoxelInputFile
void TVoronoi3D :: GetVPDInputFile()
   {
    char vpdifilename [] = "c:\\data\\ras3d.vpd";

    VPDifile = fopen(vpdifilename, "rb");
    if (VPDifile == NULL)
      {
       cerr << "Could not open VPD file !" << endl;
       exit(1);
      }
    else
      {
       cout << endl;
       cout << "The VPD file is opened" << endl;
      }
   }

// Procedure: GetVPIInputFile
void TVoronoi3D :: GetVPIInputFile(VpiType& VPI)
   {
    char vpiifilename[] = "c:\\data\\ras3d.vpi";
    VPIifile = fopen(vpiifilename, "rb");
    if (VPIifile == NULL)
      {
       cerr << "Could not open VPI file !" << endl;
       exit(1);
      }
    else
      {
       cout << endl;
       cout << "Information of  " << vpiifilename << endl;
       fread(&VPI, sizeof(VpiType), 1, VPIifile);
       int size = sizeof(VpiType);

       maxrow   = VPI.Nscanlines;
```

```
      maxcol   = VPI.Npixels;
      maxlevel = VPI.Nlevels;

      cout << "VPI file size       : " << size << endl;
      cout << endl;
      cout << "Number of lines     : " << VPI.Nscanlines << endl;
      cout << "Number of columns   : " << VPI.Npixels << endl;
      cout << "Number of levels    : " << VPI.Nlevels << endl;
      cout << "Minimum value       : " << VPI.Pvmin << endl;
      cout << "Maximum value       : " << VPI.Pvmax << endl;
      cout << "Map type            : " << VPI.Maptype << endl;
      cout << "Patched             : " << VPI.Patch << endl;
      cout << "Scale (power of 10) : " << VPI.Scale << endl;
      cout << "Coordinate type     : " << VPI.Crdtype << endl;
      cout << "Transformation parameters :" << endl;
      cout << "   a11  : " << VPI.a11 << endl;
      cout << "   a12  : " << VPI.a12 << endl;
      cout << "   a13  : " << VPI.a31 << endl;
      cout << "   a21  : " << VPI.a21 << endl;
      cout << "   a22  : " << VPI.a22 << endl;
      cout << "   a23  : " << VPI.a23 << endl;
      cout << "   a31  : " << VPI.a31 << endl;
      cout << "   a32  : " << VPI.a32 << endl;
      cout << "   a33  : " << VPI.a33 << endl;
      cout << endl;
      cout << "Translation vectors : " << endl;
      cout << "   b1   : " << VPI.b1 << endl;
      cout << "   b2   : " << VPI.b2 << endl;
      cout << "   b3   : " << VPI.b3 << endl;
      fclose(VPIifile);
     }
  }

// Procedure: ReadVoxelImage
void TVoronoi3D :: ReadVoxelImage(Voxel3D& Voxel)
  {
   int r;
   int c;
   int l;

   cout << endl;
   cout << "Reading voxels data ..., wait ! " << endl;
   cout << endl;

   for (l = 0; l < maxlevel; l ++)
     {
      for (r = 0; r < maxrow; r ++)
        {
         for (c = 0; c < maxcol; c ++)
           {
            fread(&Voxel[l][r][c], sizeof(short), 1, VPDifile);
           }
        }
     }
   fclose(VPDifile);
   cout << endl;
  }

// Procedure: CopyVoxel
void TVoronoi3D :: CopyVoxel(Voxel3D VoxA, Voxel3D& VoxB)
  {
```

```
   int i, j, k;
   for (i = 0; i < maxlevel; i ++)
     {
      for (j = 0; j < maxrow; j ++)
        {
         for (k = 0; k < maxcol; k ++)
           {
            VoxB[i][j][k] = VoxA[i][j][k];
           }
        }
     }
  }

// Procedure: PrintPixel
void TVoronoi3D :: PrintVoxel(Voxel3D Voxel)
  {
   int r;
   int c;
   int l;

   for (r = 0; r < maxrow; r ++)
     {
      for (c = 0; c < maxcol; c ++)
        {
         for (l = 0; l < maxlevel; l ++)
           {
            cout << "Voxel value at [row][col][level]: [" << r << "][" << c
                 << "][" << l << "]: " << Voxel[r][c][l] << "\r";
           }
        }
     }
  }


// Procedure: WriteVoxelOutputFile
void TVoronoi3D :: WriteVPDOutputFileDist()
  {
   int r;
   int l;
   int c;

   char vpdodfilename[] = "c:\\data\\tvoxod.vpd";
   VPDodfile = fopen(vpdodfilename, "wb");
   for (l = 0; l < maxlevel; l ++)
     {
      for (r = 0; r < maxrow; r ++)
        {
         for (c = 0; c < maxcol; c ++)
           {
            fwrite(&Voxel[l][r][c], sizeof(short), 1, VPDodfile);
           }
        }
     }
   fclose(VPDodfile);
  }

// Procedure: WriteVPIFile
void TVoronoi3D :: WriteVPIOutputFileDist()
  {
   FILE* VPIofile;
```

```
   char vpifilename[] = "c:\\data\\tvoxod.vpi";
   VPIofile = fopen(vpifilename, "wb");

   // assign the value for the VPI file
   VPIOut.Nscanlines = VPI.Nscanlines;
   VPIOut.Npixels = VPI.Npixels;
   VPIOut.Nlevels = VPI.Nlevels;
   VPIOut.Pvmin = 0;
   VPIOut.Pvmax = 200;
   VPIOut.Maptype = 2;
   VPIOut.Patch = 0;
   VPIOut.Scale = 0;
   VPIOut.Crdtype = 1;
   VPIOut.a11 = VPI.a11;
   VPIOut.a12 = 0;
   VPIOut.a13 = 0;
   VPIOut.a21 = 0;
   VPIOut.a22 = VPI.a22;
   VPIOut.a23 = 0;
   VPIOut.a31 = 0;
   VPIOut.a32 = 0;
   VPIOut.a33 = VPI.a33;
   VPIOut.b1 = 1;
   VPIOut.b2 = 1;
   VPIOut.b3 = maxlevel + 1;

   // write out the above values to the VPI file
   fwrite(&VPIOut, sizeof(VpiType), 1, VPIofile);
   fclose(VPIofile);
  }

// Procedure: WriteVPDOutputFileVoronoi
void TVoronoi3D :: WriteVPDOutputFileVoronoi()
  {
   int r;
   int l;
   int c;

   char vpdovfilename[] = "c:\\data\\tvoxov.vpd";
   VPDovfile = fopen(vpdovfilename, "wb");
   for (l = 0; l < maxlevel; l ++)
     {
      for (r = 0; r < maxrow; r ++)
        {
         for (c = 0; c < maxcol; c ++)
           {
            fwrite(&VoxelVor[l][r][c], sizeof(short), 1, VPDovfile);
           }
        }
     }
   fclose(VPDovfile);
  }

// Procedure: WriteVPIOutputFileVoronoi
void TVoronoi3D :: WriteVPIOutputFileVoronoi()
  {
   FILE* VPIovfile;
   char vpiovfilename[] = "c:\\data\\tvoxov.vpi";
   VPIovfile = fopen(vpiovfilename, "wb");

   // assign the value for the VPI file
```

```
   VPIOut.Nscanlines = VPI.Nscanlines;
   VPIOut.Npixels = VPI.Npixels;
   VPIOut.Nlevels = VPI.Nlevels;
   VPIOut.Pvmin = 0;
   VPIOut.Pvmax = 200;
   VPIOut.Maptype = 2;
   VPIOut.Patch = 0;
   VPIOut.Scale = 0;
   VPIOut.Crdtype = 1;
   VPIOut.a11 = VPI.a11;
   VPIOut.a12 = 0;
   VPIOut.a13 = 0;
   VPIOut.a21 = 0;
   VPIOut.a22 = VPI.a22;
   VPIOut.a23 = 0;
   VPIOut.a31 = 0;
   VPIOut.a32 = 0;
   VPIOut.a33 = VPI.a33;
   VPIOut.b1 = 1;
   VPIOut.b2 = 1;
   VPIOut.b3 = maxlevel + 1;

   // write out the above values to the VPI file
   fwrite(&VPIOut, sizeof(VpiType), 1, VPIovfile);
   fclose(VPIovfile);
  }


// Procedure: SetBackground
void TVoronoi3D :: SetBackground(Voxel3D Voxel, int Bg, int Fg)
  {
   int row;
   int col;
   int l;

   for (l = 0; l < maxlevel; l ++)
     {
      for (row = 0; row < maxrow; row ++)
        {
         for (col = 0; col < maxcol; col ++)
           {
            if (Voxel[l][row][col] == 0)
                Voxel[l][row][col] = Bg;
            else
            if (Fg > 0)
                Voxel[l][row][col] = Fg;
            }

        }
     }
  }

// Procedure: GetUpperMaskDist
void TVoronoi3D :: GetUpperMaskDist(int l, int r, int c, Voxel3D Voxel, Mask&
MaskPix)
  {
   MaskPix[0]  = Voxel[l-1][r-1][c-1];
   MaskPix[1]  = Voxel[l-1][r-1][c];
   MaskPix[2]  = Voxel[l-1][r-1][c+1];
   MaskPix[3]  = Voxel[l-1][r][c-1];
   MaskPix[4]  = Voxel[l-1][r][c];
```

```
   MaskPix[5]  = Voxel[l-1][r][c+1];
   MaskPix[6]  = Voxel[l-1][r+1][c-1];
   MaskPix[7]  = Voxel[l-1][r+1][c];
   MaskPix[8]  = Voxel[l-1][r+1][c+1];
   MaskPix[9]  = Voxel[l][r-1][c-1];
   MaskPix[10] = Voxel[l][r-1][c];
   MaskPix[11] = Voxel[l][r-1][c+1];
   MaskPix[12] = Voxel[l][r][c-1];
   MaskPix[13] = Voxel[l][r][c];

  }

// Procedure: GetLowerMaskDist
void TVoronoi3D :: GetLowerMaskDist(int l, int r, int c, Voxel3D Voxel, Mask&
MaskPix)
  {
   MaskPix[13] = Voxel[l][r][c];
   MaskPix[14] = Voxel[l][r][c+1];
   MaskPix[15] = Voxel[l][r+1][c-1];
   MaskPix[16] = Voxel[l][r+1][c];
   MaskPix[17] = Voxel[l][r+1][c+1];
   MaskPix[18] = Voxel[l+1][r-1][c-1];
   MaskPix[19] = Voxel[l+1][r-1][c];
   MaskPix[20] = Voxel[l+1][r+1][c+1];
   MaskPix[21] = Voxel[l+1][r][c-1];
   MaskPix[22] = Voxel[l+1][r][c];
   MaskPix[23] = Voxel[l+1][r][c+1];
   MaskPix[24] = Voxel[l+1][r+1][c-1];
   MaskPix[25] = Voxel[l+1][r+1][c];
   MaskPix[26] = Voxel[l+1][r+1][c+1];

  }

// Procedure: GetUpperMaskVoronoi
void TVoronoi3D :: GetUpperMaskVoronoi(int l, int r, int c, Voxel3D VoxelVor,
Mask& MaskPixVor)
  {
   MaskPixVor[0]  = VoxelVor[l-1][r-1][c-1];
   MaskPixVor[1]  = VoxelVor[l-1][r-1][c];
   MaskPixVor[2]  = VoxelVor[l-1][r-1][c+1];
   MaskPixVor[3]  = VoxelVor[l-1][r][c-1];
   MaskPixVor[4]  = VoxelVor[l-1][r][c];
   MaskPixVor[5]  = VoxelVor[l-1][r][c+1];
   MaskPixVor[6]  = VoxelVor[l-1][r+1][c-1];
   MaskPixVor[7]  = VoxelVor[l-1][r+1][c];
   MaskPixVor[8]  = VoxelVor[l-1][r+1][c+1];
   MaskPixVor[9]  = VoxelVor[l][r-1][c-1];
   MaskPixVor[10] = VoxelVor[l][r-1][c];
   MaskPixVor[11] = VoxelVor[l][r-1][c+1];
   MaskPixVor[12] = VoxelVor[l][r][c-1];
   MaskPixVor[13] = VoxelVor[l][r][c];
  }

// Procedure: GetLowerMaskVoronoi
void TVoronoi3D :: GetLowerMaskVoronoi(int l, int r, int c, Voxel3D VoxelVor,
Mask& MaskPixVor)
  {
   MaskPixVor[13] = VoxelVor[l][r][c];
   MaskPixVor[14] = VoxelVor[l][r][c+1];
   MaskPixVor[15] = VoxelVor[l][r+1][c-1];
   MaskPixVor[16] = VoxelVor[l][r+1][c];
```

```
   MaskPixVor[17] = VoxelVor[l][r+1][c+1];
   MaskPixVor[18] = VoxelVor[l+1][r-1][c-1];
   MaskPixVor[19] = VoxelVor[l+1][r-1][c];
   MaskPixVor[20] = VoxelVor[l+1][r+1][c+1];
   MaskPixVor[21] = VoxelVor[l+1][r][c-1];
   MaskPixVor[22] = VoxelVor[l+1][r][c];
   MaskPixVor[23] = VoxelVor[l+1][r][c+1];
   MaskPixVor[24] = VoxelVor[l+1][r+1][c-1];
   MaskPixVor[25] = VoxelVor[l+1][r+1][c];
   MaskPixVor[26] = VoxelVor[l+1][r+1][c+1];
  }

// Function: MinByIndex
int TVoronoi3D :: MinByIndex(int from, int to)
  {
   int i;
   int temp;
   int Idx;

   temp = MaskPix[from];
   Idx = from;
   for (i = from; i <= to ; i ++)
     {
      if (temp > MaskPix[i])
        {
         temp = MaskPix[i];
         Idx = i;
        }
     }
   return Idx;
  }

#define min(a, b) (((a) < (b)) ? (a) : (b))

// Function Minimum of 5 values
int TVoronoi3D :: Min5(int a, int b, int c, int d, int e)
  {
   int mn;

   mn = min(a, b);
   mn = min(mn, c);
   mn = min(mn, d);
   mn = min(mn, e);
   return mn;
  }

// Procedure: ForwardPass
void TVoronoi3D :: ForwardPass(Voxel3D Voxel, Voxel3D VoxelVor)
  {
   int r;
   int c;
   int l;
   int k;

   for (l = 1; l < maxlevel-1; l ++)
     {
      for (r = 1; r < maxrow-1; r ++)
        {
         for (c = 1; c < maxcol-1; c ++)
           {
            GetUpperMaskDist(l, r, c, Voxel, MaskPix);
```

```
            GetUpperMaskVoronoi(l, r, c, VoxelVor, MaskPixVor);
            for (k = 0; k < 13; k ++)
               {
                if ((k == 0) || (k == 2) ||
                    (k == 6) || (k == 8))
                   MaskPix[k] = MaskPix[k] + 5;

                if ((k == 1) || (k == 3) ||
                    (k == 5) || (k == 7) ||
                    (k == 9) || (k == 11))
                   MaskPix[k] = MaskPix[k] + 4;

                if ((k == 4) || (k == 10) || (k == 12))
                   MaskPix[k] = MaskPix[k] + 3;
                }

            if (MaskPix[13] != 30000)
                MaskPix[13] = 0;

            Voxel[l][r][c] = MaskPix[MinByIndex(0, 13)];
            VoxelVor[l][r][c] = MaskPixVor[MinByIndex(0, 13)];
           }
        }
     }
  }

// Procedure: BackwardPass
void TVoronoi3D :: BackwardPass(Voxel3D Voxel, Voxel3D VoxelVor)
  {
   int r;
   int c;
   int l;
   int k;

   for (l = maxlevel - 2; l > 0; l --)
     {
      for (r = maxrow - 2; r > 0; r --)
        {
         for (c = maxcol - 2; c > 0; c --)
            {
             GetLowerMaskDist(l, r, c, Voxel, MaskPix);
             GetLowerMaskVoronoi(l, r, c, VoxelVor, MaskPixVor);
             for (k = 26; k > 13; k --)
               {
                 if ((k == 18) || (k == 20) ||
                     (k == 24) || (k == 26))
                    MaskPix[k] = MaskPix[k] + 5;

                 if ((k == 19) || (k == 21) ||
                     (k == 23) || (k == 25) ||
                     (k == 15) || (k == 17))
                    MaskPix[k] = MaskPix[k] + 4;

                 if ((k == 14) || (k == 16) || (k == 22))
                    MaskPix[k] = MaskPix[k] + 3;
               }
             Voxel[l][r][c] = MaskPix[MinByIndex(13, 26)];
             VoxelVor[l][r][c] = MaskPixVor[MinByIndex(13, 26)];
             }
        }
     }
```

```
   }

// Procedure: ForwardVoronoi
void TVoronoi3D :: ForwardVoronoi()
  {
   cout << "D3 Forward pass ..." << endl;
   ReadVoxelImage(Voxel);
   CopyVoxel(Voxel, VoxelVor);
   SetBackground(Voxel, 30000, 0);
   ForwardPass(Voxel, VoxelVor);
  }

// Procedure: BackwardVoronoi
void TVoronoi3D :: BackwardVoronoi()
  {
   cout << "3D Backward pass ..." << endl;
   BackwardPass(Voxel, VoxelVor);
  }

// Procedure: Title
void TVoronoi3D :: Title()
  {
   cout << endl;
   cout << "----------------- Object-Oriented 3D DT and Voronoi Tessellation
--------------" << endl;
   cout << "----------- By: Alias Abdul-Rahman, November, 1998 ------------"
<< endl;
   cout << "   Input/Output : 3D array ILWIS look-like file format. " << endl;
   cout << endl;
  }
```

3D TIN Data Structuring Code

The following gives the complete code for the 3D TIN (or TEN) data structure generation program.

```
// ************************************************************************
//
//                      Object-Oriented 3D Voronoi-to-3D TIN (TEN) program
//
//              Input  :  VPI and VPD file of 3D Voronoi program
//
//              Output :  3D TIN (TEN) table (Ascii file)
//
//                        TEN = Tetrahedron Network
//
// ************************************************************************
//

#include <iostream.h>
#include <stdlib.h>
#include <stdio.h>
#include <fstream.h>
#include <io.h>
#include <iomanip.h>
#include <dos.h>
#include <alloc.h>
#include <malloc.h>
#include <conio.h>
#include <limits.h>
#include <stddef.h>

#include "TVor3D.h"

#define masksize 8           // 2 x 2 x 2 Mask size

class TTinGen3D : public TVoronoi3D
  {
   public:
        TTinGen3D();
        ~TTinGen3D();

        typedef short DataType;
        typedef struct VertexStruct
                  {
                   DataType N1;
                   DataType N2;
                   DataType N3;
                   DataType N4;
                  } TVertex;

        typedef struct TsNodeStruct
                  {
```

```
                short x;
                short y;
              } TsNode;

      typedef short Image;
      typedef Image* VoxelRow;
      typedef VoxelRow* Voxel2D;
      typedef Voxel2D* Voxel3D;
      typedef short Mask[8];

      // Global variables
      Mask MaskPix;
      Voxel3D Voxel;
      FILE* VPDfile;
      FILE* VPIfile;

      FILE* TENfile;
      int nTEN;
      TVertex Element;
      TsNode TsPnt;
      bool FoundTEN;

      // Function members
      void GetVPDfile();
      void ReadVoxelImage(Voxel3D&);
      void GetVPIfile(VpiStruct&);
      void Get3DTINfile();
      bool Less(DataType, DataType);
      bool Greater(DataType, DataType);
      void Swap(DataType&, DataType&);
      void NodeOrder(DataType&, DataType&, DataType&, DataType&);
      void AddTENtoFile(TVertex);
      void GetMaskVox(int, int, int, Voxel3D, Mask&);
      void GetSubImage(int, int, int, Voxel3D, Mask&);
      void DetectTEN(Mask);
      void ScanVoxels(Voxel3D);
      void Make3DTIN();
      void Title();
      void PressAnyKey();
};
```

```
// Object-Oriented 3D TIN Tessellation Program
// File: TVorTin3D.cpp

#include "TTinGen3D.h"


// main program
void main()
  {
   TTinGen3D VoxTEN;
   VoxTEN.Title();
   TTinGen3D();
   VoxTEN.GetVPDfile();
   VoxTEN.Get3DTINfile();
   VoxTEN.Make3DTIN();
   VoxTEN.PressAnyKey();
  }

// Definitions section
// The constructor
TTinGen3D :: TTinGen3D()
  {
   GetVPIfile(VPI);

   // Allocate memory dynamically
   Voxel = new Voxel2D[maxlevel];
   for (l = 0; l < maxlevel; l ++)
     {
      Voxel[l] = new VoxelRow[maxrow];
      for (r = 0; r < maxrow; r ++)
        {
         Voxel[l][r] = new Image[maxcol];
        }
     }
  }

TTinGen3D :: ~TTinGen3D()
  {
   // Deallocate memory
   for (l = 0; l < maxlevel; l ++)
     {
      for (r = 0; r < maxrow; r ++)
        {
         delete [] Voxel[l][r];
        }
     }
   delete [] Voxel;
  }

// Procedure: PresAnyKey
void TTinGen3D :: PressAnyKey()
  {
   cout << endl;
   cout << "Press any key to continue" << endl;
   getch();
   cout << endl;
  }

// Procedure: GetMPDFile
void TTinGen3D :: GetVPDfile()
  {
```

```
   char vpdfilename [] = "c:\\data\\pnt24v.vpd";
   VPDfile = fopen(vpdfilename, "rb");

   if (VPDfile == NULL)
     {
      cerr << "Could not open VPD file !" << endl;
      exit(1);
     }
   else
     {
      cout << endl;
      cout << "The VPD file is opened" << endl;
     }
  }

void TTinGen3D :: ReadVoxelImage(Voxel3D& Voxel)
  {
   int l, r, c;
   for (l = 0; l < maxlevel; l ++)
     {
      for (r = 0; r < maxrow; r++)
        {
         for (c = 0; c < maxcol; c ++)
           {
            fread(&Voxel[l][r][c], sizeof(short), 1, VPDfile);
           }
        }
     }
   cout << endl;
  fclose(VPDfile);
  }

void TTinGen3D :: GetVPIfile(VpiStruct& VPI)
  {
   char vpifilename [] = "c:\\data\\pnt24v.vpi";
   VPIfile = fopen(vpifilename, "rb");

   if (VPIfile == NULL)
     {
      cerr << "Could not open VPI file !" << endl;
      exit(1);
     }
   else
     {
      cout << "Information of  " << vpifilename << endl;
      fread(&VPI, sizeof(VpiType), 1, VPIfile);
      int size = sizeof(VpiType);

      maxrow = VPI.Nscanlines;
      maxcol = VPI.Npixels;
      maxlevel = VPI.Nlevels;

      cout << "MPI file size       : " << size << endl;
      cout << endl;
      cout << "Number of lines     : " << VPI.Nscanlines << endl;
      cout << "Number of columns   : " << VPI.Npixels << endl;
      cout << "Number of levels    : " << VPI.Nlevels << endl;
      cout << "Minimum value       : " << VPI.Pvmin << endl;
      cout << "Maximum value       : " << VPI.Pvmax << endl;
      cout << "Map type            : " << VPI.Maptype << endl;
      cout << "Patched             : " << VPI.Patch << endl;
```

```
      cout << "Scale (power of 10) : " << VPI.Scale << endl;
      cout << "Coordinate type     : " << VPI.Crdtype << endl;
      cout << "Transformation parameters :" << endl;
      cout << "   a11  : " << VPI.a11 << endl;
      cout << "   a12  : " << VPI.a12 << endl;
      cout << "   a13  : " << VPI.a13 << endl;
      cout << "   a21  : " << VPI.a21 << endl;
      cout << "   a22  : " << VPI.a22 << endl;
      cout << "   a23  : " << VPI.a23 << endl;
      cout << "   a31  : " << VPI.a31 << endl;
      cout << "   a32  : " << VPI.a32 << endl;
      cout << "   a33  : " << VPI.a33 << endl;
      cout << "Translation vectors: " << endl;
      cout << "   b1   : " << VPI.b1 << endl;
      cout << "   b2   : " << VPI.b2 << endl;
      cout << "   b3   : " << VPI.b3 << endl;
      fclose(VPIfile);
     }
  }

// Procedure: Get3DTINfile
void TTinGen3D :: Get3DTINfile()
  {
   char TENfilename [] = "c:\\data\\pnt24.ten";
   TENfile = fopen(TENfilename, "w");

   char* charN1 = "Node1";
   char* charN2 = "Node2";
   char* charN3 = "Node3";
   char* charN4 = "Node4";

   fprintf(TENfile, "%8s %8s %8s %8s\n", charN1, charN2, charN3, charN4);
   if (TENfile == NULL)
     {
      cerr << "Could not open 3D TIN file !" << endl;
      exit(1);
     }
   else
     {
      cout << endl;
      cout << "The 3D TIN file is okay for writing ... " << endl;
     }
  }

// Function: Less
bool TTinGen3D :: Less(DataType a, DataType b)
  {
   return (a < b);
  }

// Function: Greater
bool TTinGen3D :: Greater(DataType a, DataType b)
  {
   return (a > b);
  }

//Procedure: Swap
void TTinGen3D :: Swap(DataType& a, DataType& b)
  {
   DataType temp;
```

```
   temp = a;
   a = b;
   b = temp;
  }

// Procedure: NodeOrder
void TTinGen3D :: NodeOrder(DataType& a, DataType& b, DataType& c, DataType&
d)
  {
   if (Greater(a, b))
       Swap(a, b);
   if (Greater(b, c))
       Swap(b, c);
   if (Greater(c, d))
       Swap(c, d);
   if (Greater(a, b))
       Swap(a, b);
  }

// Procedure: AddTENtoFile
void TTinGen3D :: AddTENtoFile(TVertex TEN)
  {
   fprintf(TENfile, "%8d %8d %8d %8d \n", TEN.N1, TEN.N2, TEN.N3, TEN.N4);
  }

//Procedure: GetSubImage
void TTinGen3D :: GetSubImage(int k, int i, int c, Voxel3D Voxel, Mask&
MaskPix)
  {
   int m, j;
   m = -1;
   for (k = 0; k < maxlevel-1; k ++)
     {
      for (i = 0; i < maxrow-1; i ++)
       {
        for (j = c; j < c + 1; j ++)
          {
           m ++;
           MaskPix[m] = Voxel[k][i][j];
          }
       }
     }
  }

// Procedure: GetMask
void TTinGen3D :: GetMaskVox(int l, int r, int c, Voxel3D Voxel, Mask&
MaskPix)
  {
   MaskPix[0] = Voxel[l][r][c];
   MaskPix[1] = Voxel[l][r][c+1];
   MaskPix[2] = Voxel[l][r+1][c];
   MaskPix[3] = Voxel[l][r+1][c+1];
   MaskPix[4] = Voxel[l+1][r][c];
   MaskPix[5] = Voxel[l+1][r][c+1];
   MaskPix[6] = Voxel[l+1][r+1][c];
   MaskPix[7] = Voxel[l+1][r+1][c+1];
  }

// Procedure: DEtectTEN
void TTinGen3D :: DetectTEN(Mask MaskPix)
  {
```

```
      bool DetectTEN1, DetectTEN2, DetectTEN3, DetectTEN4, DetectTEN5,
DetectTEN6;
     bool DetectA, DetectB, DetectC, DetectD, DetectE, DetectF, DetectG,
DetectH;
   bool DetectI, DetectJ, DetectK, DetectL;

   DetectTEN1 = (MaskPix[0] != MaskPix[2]) &&
                (MaskPix[2] != MaskPix[3]) &&
                (MaskPix[3] != MaskPix[4]) &&
                (MaskPix[0] != MaskPix[3]) &&
                (MaskPix[0] != MaskPix[4]) &&
                (MaskPix[2] != MaskPix[4]);

   DetectTEN2 = (MaskPix[0] != MaskPix[1]) &&
                (MaskPix[1] != MaskPix[3]) &&
                (MaskPix[3] != MaskPix[4]) &&
                (MaskPix[0] != MaskPix[3]) &&
                (MaskPix[0] != MaskPix[4]) &&
                (MaskPix[1] != MaskPix[4]);

   DetectTEN3 = (MaskPix[2] != MaskPix[3]) &&
                (MaskPix[3] != MaskPix[4]) &&
                (MaskPix[4] != MaskPix[6]) &&
                (MaskPix[2] != MaskPix[4]) &&
                (MaskPix[2] != MaskPix[6]) &&
                (MaskPix[3] != MaskPix[6]);

   DetectTEN4 = (MaskPix[1] != MaskPix[3]) &&
                (MaskPix[3] != MaskPix[4]) &&
                (MaskPix[4] != MaskPix[5]) &&
                (MaskPix[1] != MaskPix[4]) &&
                (MaskPix[1] != MaskPix[5]) &&
                (MaskPix[3] != MaskPix[5]);

   DetectTEN5 = (MaskPix[3] != MaskPix[4]) &&
                (MaskPix[4] != MaskPix[5]) &&
                (MaskPix[5] != MaskPix[7]) &&
                (MaskPix[3] != MaskPix[5]) &&
                (MaskPix[3] != MaskPix[7]) &&
                (MaskPix[4] != MaskPix[7]);

   DetectTEN6 = (MaskPix[3] != MaskPix[4]) &&
                (MaskPix[4] != MaskPix[6]) &&
                (MaskPix[6] != MaskPix[7]) &&
                (MaskPix[3] != MaskPix[6]) &&
                (MaskPix[3] != MaskPix[7]) &&
                (MaskPix[4] != MaskPix[7]);

      DetectA = (MaskPix[0] != MaskPix[5]);
      DetectB = (MaskPix[0] != MaskPix[6]);
      DetectC = (MaskPix[0] != MaskPix[7]);
      DetectD = (MaskPix[1] != MaskPix[2]);
      DetectE = (MaskPix[1] != MaskPix[6]);
      DetectF = (MaskPix[1] != MaskPix[7]);
      DetectG = (MaskPix[2] != MaskPix[5]);
      DetectH = (MaskPix[2] != MaskPix[7]);
      DetectI = (MaskPix[5] != MaskPix[6]);

     DetectJ = (! ((MaskPix[0] != MaskPix[4]) && (MaskPix[0] == MaskPix[3])
                                                  && (MaskPix[4] ==
MaskPix[7])));
```

```
      DetectK = (! ((MaskPix[2] != MaskPix[3]) && (MaskPix[2] == MaskPix[4])
                                                    && (MaskPix[3] ==
MaskPix[5])));


      DetectL = (! ((MaskPix[1] != MaskPix[3]) && (MaskPix[1] == MaskPix[4])
                                                    && (MaskPix[3] ==
MaskPix[6])));



   nTEN = 0;
   if (DetectTEN1 && DetectB && DetectC && DetectD
                  && DetectE && DetectG && DetectL)

       {
        Element.N1 = MaskPix[0];
        Element.N2 = MaskPix[2];
        Element.N3 = MaskPix[3];
        Element.N4 = MaskPix[4];
        nTEN ++;
        AddTENtoFile(Element);
        FoundTEN = true;
       }

   else
   if (DetectTEN2 && DetectA && DetectC && DetectD
                  && DetectE && DetectG && DetectK)

       {
        Element.N1 = MaskPix[0];
        Element.N2 = MaskPix[1];
        Element.N3 = MaskPix[3];
        Element.N4 = MaskPix[4];
        nTEN ++;
        AddTENtoFile(Element);
        FoundTEN = true;
       }
   else
   if (DetectTEN3 && DetectB && DetectC && DetectE
                  && DetectG && DetectH && DetectJ)

       {
        Element.N1 = MaskPix[2];
        Element.N2 = MaskPix[3];
        Element.N3 = MaskPix[4];
        Element.N4 = MaskPix[6];
        nTEN ++;
        AddTENtoFile(Element);
        FoundTEN = true;
       }
    else
    if (DetectTEN4 && DetectA && DetectC && DetectE
                   && DetectF && DetectG && DetectJ)

       {
        Element.N1 = MaskPix[1];
        Element.N2 = MaskPix[3];
        Element.N3 = MaskPix[4];
        Element.N4 = MaskPix[5];
        nTEN ++;
```

```
        AddTENtoFile(Element);
        FoundTEN = true;
       }
    else
    if (DetectTEN5 && DetectC && DetectE && DetectG
                   && DetectH && DetectI && DetectL)

       {
        Element.N1 = MaskPix[3];
        Element.N2 = MaskPix[4];
        Element.N3 = MaskPix[5];
        Element.N4 = MaskPix[7];
        nTEN ++;
        AddTENtoFile(Element);
        FoundTEN = true;
       }
    else
    if (DetectTEN6 && DetectC && DetectE && DetectG
                   && DetectH && DetectI && DetectK)

       {
        Element.N1 = MaskPix[3];
        Element.N2 = MaskPix[4];
        Element.N3 = MaskPix[6];
        Element.N4 = MaskPix[7];
        nTEN ++;
        AddTENtoFile(Element);
        FoundTEN = true;
       }
    else
    FoundTEN = false;
  }


// Procedure: ScanVoxels
void TTinGen3D :: ScanVoxels(Voxel3D Voxel)
  {
   int l, r, c;
   for (l = 1; l < maxlevel-1; l ++)
     {
      for (r = 1; r < maxrow-1; r ++)
        {
          for (c = 1; c < maxcol-1; c ++)
            {
             GetMaskVox(l, r, c, Voxel, MaskPix);
             DetectTEN(MaskPix);
            }
        }
     }
  }

// Procedure: Make3DTIN
void TTinGen3D :: Make3DTIN()
  {
   cout << "Reading the data ..., wait !" << endl;
   ReadVoxelImage(Voxel);
   cout << "Making 3D TIN structure ..." << endl;
   ScanVoxels(Voxel);
   cout << endl;
   cout << "Writing 3D TIN table to file ... " << endl;
   fclose(TENfile);
```

```
  }

//Procedure: Title
void TTinGen3D :: Title()
  {
   cout << endl;
    cout << "-------------Object-Oriented 3D Voronoi-to-3D TIN --------" <<
endl;
    cout << "----------- By: Alias Abdul-Rahman, July, 1998 -----------" <<
endl;
   cout << "           Input : .vpd, .vpi files. " << endl;
   cout << "          Output: 3D TIN file (.ten), an ASCII file. " << endl;
   cout << endl;
  }
```

```
// tinwin2d.rh

#define IDI_ICON1 1
#define CM_CLEAR           1125
#define CM_FILEEXIT        1126
#define CM_ABOUT           1127
#define CM_INPUTFILESITEM1    18874 // for XYZ file
#define CM_POPUPITEM1      18873 // for TIN file
```

```
//******************************************************//
// A Windows program for TIN display                    //
//   Input: XYZ coordinates and TIN file                //
//   Output: TINs in Windows environment                //
//******************************************************//

#include <owl/pch.h>
#include <owl/applicat.h>
#include <owl/framewin.h>
#include <owl/decframe.h>
#include <owl/statusba.h>
#include <owl/controlb.h>
#include <owl/buttonga.h>
#include <owl/gdiobjec.h>
#include <owl/chooseco.h>
#include <owl/inputdia.h>
#include <owl/opensave.h>
#include <owl/dc.h>
#include <mem.h>

#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <stdio.h>
#include <stddef.h>
#include <string.h>

#include "tinwin2d.rh"

#define maxtriangle 20000
#define maxpoint 8000
#define maxarc 200

class TTINDrawApp : public TApplication
  {
   public:
      TTINDrawApp() : TApplication() {}  // call base class constructor
      void InitMainWindow();
  };

class TTINWindow : public TWindow
  {
   public:
      TTINWindow(TWindow* parent = 0)
         : TWindow(parent) {}

      struct Point
        {
         float x;
         float y;
         float z;
        };

      struct Triangle
        {
         int Node1;
         int Node2;
         int Node3;
        };

      Point* pnt_ptr[maxpoint];
```

```
      Triangle* tri_ptr[maxtriangle];

      struct Polygon
        {
         int Snode;
         int Enode;
        };

      Polygon* poly_ptr[maxarc];

      FILE* XYZfile;
      FILE* TINfile;
      FILE* ARCfile;

      TOpenSaveDialog :: TData fileData;
      int k, i, t, ii;
      float xmin, xmax, ymin, ymax;
      float zmin, zmax;
      float xlength, ylength;
      int npnt, ntri, narc;
      int MaxX, MaxY;
      float scaleX, scaleY;

      void ReadXYZ();
      void ReadTIN();
      void ReadARC();
      void GetMinMax(float&, float&,
                     float&, float&,
                     float&, float&);
      void Paint(TDC& dc, bool, TRect&);
      void EvSize(uint, TSize&);
      void CmInputFilesItem1();
      void CmPopupItem1();
      void CmFileExit()
           {PostQuitMessage(0);}
      void CmClear();
      void CmAbout();
      DECLARE_RESPONSE_TABLE(TTINWindow);

  };

DEFINE_RESPONSE_TABLE1(TTINWindow, TWindow)
  EV_WM_SIZE,
  EV_COMMAND(CM_INPUTFILESITEM1, CmInputFilesItem1),
  EV_COMMAND(CM_POPUPITEM1, CmPopupItem1),
  EV_COMMAND(CM_ABOUT, CmAbout),
  EV_COMMAND(CM_FILEEXIT, CmFileExit),
  EV_COMMAND(CM_CLEAR, CmClear),
END_RESPONSE_TABLE;


void TTINWindow :: CmInputFilesItem1()
  {
   ReadXYZ();
  }

void TTINWindow :: CmPopupItem1()
  {
   ReadTIN();
  }
```

```
void TTINWindow :: ReadXYZ()
  {
   string charX, charY, charZ;

   // Allocate memory for XYZ points
   for (i = 0; i < maxpoint; i ++)
     {
      pnt_ptr[i] = new Point;
     }

   // Reads again the the XYZ file
   //ifstream XYZfile = "c:\\data\\lochgrd.xyz";
   //ifstream XYZfile = "c:\\data\\lochass.xyz";
   //ifstream XYZfile = "c:\\data\\ard.xyz";
   ifstream XYZfile = "c:\\data\\pent.xyz";

   if (! XYZfile)
     {
      MessageBox ("Unable to open file: XYZ file", "FileError",
                  MB_OK | MB_ICONEXCLAMATION);
     }
   else
     {
      // reads the file header
      XYZfile >> charX >> charY >> charZ;
      npnt = 0;
        do
         {
          npnt ++;
          XYZfile >> pnt_ptr[npnt] -> x
                  >> pnt_ptr[npnt] -> y
                  >> pnt_ptr[npnt] -> z;

         } while (! XYZfile.eof());

     //if (XYZfile.eof() == true)
        //MessageBox ("Finished read the XYZ points", "XYZ file complete",
                  //MB_OK | MB_ICONINFORMATION);
     }
   XYZfile.close();
  }

void TTINWindow :: ReadTIN()
  {
   string charNode1, charNode2, charNode3;

   // Allocate memory for Triangle
   for (t = 0; t < maxtriangle; t ++)
     {
      tri_ptr[t] = new Triangle;
     }

   // Reads the TIN file
   //ifstream TINfile = "c:\\data\\lochgrd.tin";
   //ifstream TINfile = "c:\\data\\lochas2.tin";
   //ifstream TINfile = "c:\\data\\ard.tin";
   ifstream TINfile = "c:\\data\\pent.tin";


   if (!TINfile)
     {
```

```
      MessageBox("Unable to open file: TIN file",
                 "File Error", MB_OK | MB_ICONEXCLAMATION);
     }

   else
     {
      ntri = 0;

      // reads the file header
      TINfile >> charNode1 >> charNode2 >> charNode3;
      do
        {
         ntri ++;
         TINfile >> tri_ptr[ntri] -> Node1
                 >> tri_ptr[ntri] -> Node2
                 >> tri_ptr[ntri] -> Node3;

        } while (! TINfile.eof());

       //if (TINfile.eof() == true)
        //MessageBox ("Finished read the TIN nodes", "TIN file complete",
                   //MB_OK | MB_ICONINFORMATION);
      }

   TINfile.close();
  }

void TTINWindow :: ReadARC()
  {
   string charSnode, charEnode;

   // Allocate memory for XYZ points
   for (int ii = 0; ii < maxarc; ii ++)
     {
      poly_ptr[ii] = new Polygon;
     }

   // Reads again the the XYZ file
   ifstream ARCfile = "c:\\data\\lake.arc";


   if (! ARCfile)
     {
      MessageBox ("Unable to open file: ARC file", "FileError",
                  MB_OK | MB_ICONEXCLAMATION);
     }
   else
     {
      // reads the file header
      ARCfile >> charSnode >> charEnode;
      narc = 0;
        do
         {
          narc ++;
          ARCfile >> poly_ptr[narc] -> Snode
                  >> poly_ptr[narc] -> Enode;

         } while (! ARCfile.eof());

     //if (ARCfile.eof() == true)
        //MessageBox ("Finished read the ARC points", "ARC file complete",
```

```
                    //MB_OK | MB_ICONINFORMATION);
      }
    ARCfile.close();
   }


void TTINWindow :: CmClear()
   {
    // to clear and redraw
    Invalidate();
   }


void TTINWindow :: CmAbout()
   {
    MessageBox ("TINs display with menu - by Alias Abdul-Rahman, 1999",
                "About the Program", MB_OK | MB_ICONINFORMATION);
   }

void TTINWindow :: GetMinMax(float& xmax, float& ymax,
                             float& xmin, float& ymin,
                             float& xlength, float& ylength)
   {
    string charX, charY, charZ;

    // Read the XYZ file and get the min-max of the coordinates
    //ifstream XYZfile = "c:\\data\\lochgrd.xyz";
    //ifstream XYZfile = "c:\\data\\lochass.xyz";
    //ifstream XYZfile = "c:\\data\\ard.xyz";
    ifstream XYZfile = "c:\\data\\pent.xyz";

    if (! XYZfile)
       MessageBox ("Unable to open file: XYZ file", "FileError",
                   MB_OK | MB_ICONEXCLAMATION);

    float x, y, z;

    // reads the file header
    XYZfile >> charX >> charY >> charZ;

    XYZfile >> x >> y >> z;
       xmin = x;
       xmax = x;

       ymin = y;
       ymax = y;

       zmin = z;
       zmax = z;

       do
         {
          XYZfile >> x >> y >> z;

          if (xmin > x)
              xmin = x;

          if (ymin > y)
              ymin = y;

          if (zmin > z)
```

```
            zmin = z;

        if (xmax < x)
            xmax = x;

        if (ymax < y)
            ymax = y;

        if (zmax < z)
            zmax = z;
        } while (! XYZfile.eof());

      xlength = xmax - xmin;
      ylength = ymax - ymin;
  }

void TTINWindow :: Paint(TDC& dc, bool, TRect&)
  {
   GetMinMax(xmax, ymax, xmin, ymin, xlength, ylength);

   ReadXYZ(); // reads the XYZ
   ReadTIN(); // reads the TIN
   //ReadARC(); // reads the ARC

   // to set the background color of the client area
   // comment out the next line for default colour
   //static const TColor color (RGB(0, 0, 0));      // RGB (0, 0, 0) - black
   //TWindow :: SetBkgndColor(color);               // RGB (192, 192, 192) -
gray


   RECT rect;
   :: GetClientRect(HWindow, &rect);

   dc.SetMapMode(MM_ANISOTROPIC);

   scaleX = (rect.right - rect.left) / xlength;
   scaleY = (rect.bottom - rect.top) / ylength;

   for (k = 1; k < npnt; k ++)
     {
      pnt_ptr[k] -> x = (pnt_ptr[k] -> x - xmin) * scaleX;
      pnt_ptr[k] -> y = (rect.bottom - rect.top) - ((pnt_ptr[k] -> y - ymin)
                                                    * scaleY);
     }

   // set pen for points display
   TPen pen1 = TPen(RGB(255, 0, 0), 1, PS_SOLID);
   SelectObject(dc, pen1);
   for (k = 1; k < npnt; k ++)
     {
      dc.MoveTo(pnt_ptr[k] -> x, pnt_ptr[k] -> y);
      dc.LineTo(pnt_ptr[k] -> x, pnt_ptr[k] -> y);
     }

   SelectObject(dc, pen1);
   DeleteObject(pen1);


   // set pen style for TINs display
   TPen pen2 = TPen(RGB(0, 0, 255), 1, PS_SOLID);
```

```
   SelectObject(dc, pen2);

   for (k = 1; k < ntri; k ++)
     {
      dc.MoveTo(pnt_ptr[tri_ptr[k] -> Node1] -> x,
                pnt_ptr[tri_ptr[k] -> Node1] -> y);

      dc.LineTo(pnt_ptr[tri_ptr[k] -> Node2] -> x,
                pnt_ptr[tri_ptr[k] -> Node2] -> y);
      dc.LineTo(pnt_ptr[tri_ptr[k] -> Node3] -> x,
                pnt_ptr[tri_ptr[k] -> Node3] -> y);
      dc.LineTo(pnt_ptr[tri_ptr[k] -> Node1] -> x,
                pnt_ptr[tri_ptr[k] -> Node1] -> y);
     }
   SelectObject(dc, pen2);
   DeleteObject(pen2);



   // set pen style for polygon (i.e. constrained edges) display
   TPen pen3 = TPen(RGB(255, 0, 0), 2, PS_SOLID);
   SelectObject(dc, pen3);

   for (int r = 1; r < narc; r ++)
     {
      dc.MoveTo(pnt_ptr[poly_ptr[r] -> Snode] -> x,
                pnt_ptr[poly_ptr[r] -> Snode] -> y);
      dc.LineTo(pnt_ptr[poly_ptr[r] -> Enode] -> x,
                pnt_ptr[poly_ptr[r] -> Enode] -> y);
      dc.LineTo(pnt_ptr[poly_ptr[r] -> Snode] -> x,
                pnt_ptr[poly_ptr[r] -> Snode] -> y);
     }
   SelectObject(dc, pen3);
   DeleteObject(pen3);



   // Deallocate memory
   for (i = 0; i < maxpoint; i ++)
     {
      delete [] pnt_ptr[i];
     }

   for (t = 0; t < maxtriangle; t ++)
     {
      delete [] tri_ptr[t];
     }

   for (ii = 0; ii < maxarc; ii ++)
     {
      delete [] poly_ptr[ii];
     }

  }

// Force the window to repaint if resize
void TTINWindow :: EvSize(uint, TSize&)
  {
   :: InvalidateRect(HWindow, 0, true);
  }
```

```
void TTINDrawApp :: InitMainWindow()
  {
   // construct the decorated MDI frame window
   TDecoratedFrame* frame = new TDecoratedFrame(0,
                      "TINSoft ver. 1.0", new TTINWindow);

   // construct the status bar
   TStatusBar* sb = new TStatusBar(frame, TGadget :: Recessed,
                                     TStatusBar :: CapsLock | TStatusBar ::
NumLock);

   // construct the control bar
   TControlBar* cb = new TControlBar(frame);

   cb -> Insert(*new TButtonGadget(CM_INPUTFILESITEM1, CM_INPUTFILESITEM1,
                    TButtonGadget :: Command));
   cb -> Insert(*new TButtonGadget(CM_POPUPITEM1, CM_POPUPITEM1,
                    TButtonGadget :: Command));

   cb -> Insert(*new TSeparatorGadget);

   cb -> Insert(*new TButtonGadget(CM_CLEAR, CM_CLEAR,
                    TButtonGadget :: Command));

   cb -> Insert(*new TSeparatorGadget);

   cb -> Insert(*new TButtonGadget(CM_FILEEXIT, CM_FILEEXIT,
                    TButtonGadget :: Command));

   cb -> Insert(*new TSeparatorGadget);

   cb -> Insert(*new TButtonGadget(CM_ABOUT, CM_ABOUT,
                    TButtonGadget :: Command));

   cb -> SetHintMode(TGadgetWindow :: EnterHints);

   // set client area to the application workspace
   frame -> SetBkgndColor(::GetSysColor(COLOR_APPWORKSPACE));

   // insert the status and control bar into the frame
   frame -> Insert(*sb, TDecoratedFrame :: Bottom);
   frame -> Insert(*cb, TDecoratedFrame :: Top);

   // set the main window and its menu
   SetMainWindow(frame);
   GetMainWindow() -> AssignMenu("COMMANDS");
   EnableCtl3d(true);
  }


int OwlMain(int /*argc*/, char* /*argv*/ [])
  {
   return TTINDrawApp().Run();
  }
```

```
#include "tinwin2d.rh"

COMMANDS MENU
{
 POPUP "&InputFiles"
 {
  MENUITEM "&XYZ File", CM_INPUTFILESITEM1
  MENUITEM "&TIN File", CM_POPUPITEM1
 }

 POPUP "&Redraw"
 {
  MENUITEM "&Clear and Redraw", CM_CLEAR
  MENUITEM "E&xit", CM_FILEEXIT
 }

 MENUITEM "&About", CM_ABOUT
}

STRINGTABLE
  {
   CM_INPUTFILESITEM1,  "Get the XYZ file"
   CM_POPUPITEM1,       "Get the TIN file"
   CM_CLEAR,            "Redraw the TINs"
   CM_FILEEXIT,         "Exit the program ..."
   CM_ABOUT,            "About the program ..."
  }

CM_CLEAR BITMAP
{
 '42 4D 66 01 00 00 00 00 00 00 76 00 00 00 28 00'
 '00 00 14 00 00 00 14 00 00 00 01 00 04 00 00 00'
 '00 00 F0 00 00 00 00 00 00 00 00 00 00 00 00 00'
 '00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 80'
 '00 00 00 80 80 00 80 00 00 00 80 00 80 00 80 80'
 '00 00 C0 C0 C0 00 80 80 80 00 00 00 FF 00 00 FF'
 '00 00 00 FF FF 00 FF 00 00 00 FF 00 FF 00 FF FF'
 '00 00 FF FF FF 00 91 77 77 77 77 77 77 77 77 99'
 '00 00 79 77 77 77 77 77 77 77 79 17 00 00 77 17'
 '77 87 77 77 77 79 91 77 00 00 77 77 80 08 88 87'
 '77 77 77 77 00 00 77 78 00 00 08 88 77 77 77 77'
 '00 00 77 78 0A 0A A0 08 88 87 77 77 00 00 77 78'
 '0A 0A AA A0 88 88 77 77 00 00 77 80 AA 0A AA AA'
 '00 88 77 77 00 00 77 80 AA A0 AA A0 A0 88 77 77'
 '00 00 77 0A AA A0 AA 0A A0 88 71 17 00 00 77 0A'
 'AA A0 00 AA A0 88 79 97 00 00 78 0A AA A0 AA AA'
 'A0 88 77 77 00 00 78 00 00 00 AA AA A0 88 77 77'
 '00 00 78 88 87 78 0A AA A0 88 77 77 00 00 77 77'
 '77 77 80 AA A0 88 77 77 00 00 77 71 77 77 78 0A'
 'A0 88 77 77 00 00 77 19 77 77 77 80 A0 88 77 77'
 '00 00 77 97 77 77 77 78 00 88 79 77 00 00 71 77'
 '77 77 77 77 88 87 71 19 00 00 19 77 77 77 77 77'
 '77 77 77 71 00 00'
}


CM_FILEEXIT BITMAP
{
 '42 4D 66 01 00 00 00 00 00 00 76 00 00 00 28 00'
 '00 00 14 00 00 00 14 00 00 00 01 00 04 00 00 00'
```

```
 '00 00 F0 00 00 00 00 00 00 00 00 00 00 00 00'
 '00 00 00 00 00 00 00 00 00 00 00 80 00 00 80'
 '00 00 00 80 80 00 80 00 00 00 80 00 80 00 80 80'
 '00 00 C0 C0 C0 00 80 80 80 00 00 00 FF 00 00 FF'
 '00 00 00 FF FF 00 FF 00 00 00 FF 00 FF 00 FF FF'
 '00 00 FF FF FF 00 EE EE EE EE EE EE EE EE EE EE'
 '00 00 EE EE FF EF EF FE EE EE EE EE 00 00 EE EE'
 'FF EE FE EF EE EE EE EE 00 00 EE EE FF EF FF FF'
 'EE EE EE EE 00 00 EE EE EF EE EF FF EE EE EE EE'
 '00 00 88 88 88 88 88 88 88 88 88 88 00 00 66 66'
 '8A AA A8 77 86 66 66 66 00 00 66 66 8A AA A8 77'
 '86 66 66 66 00 00 66 66 8A AA A8 77 86 66 66 66'
 '00 00 66 66 8A A9 A8 77 86 66 66 66 00 00 66 66'
 '8A A9 A8 77 86 66 66 66 00 00 66 66 8A AA A8 77'
 '86 66 66 66 00 00 66 66 8A AA A8 77 86 66 66 66'
 '00 00 66 66 8A AA A8 77 86 66 66 66 00 00 66 66'
 '8A A8 87 77 86 66 66 66 00 00 66 66 8A 87 77 77'
 '86 66 66 66 00 00 66 66 88 77 77 77 86 66 66 66'
 '00 00 66 66 88 88 88 88 86 66 66 66 00 00 66 66'
 '66 66 66 66 66 66 66 66 00 00 66 66 66 66 66 66'
 '66 66 66 66 00 00'
}


CM_ABOUT BITMAP
{
 '42 4D 66 01 00 00 00 00 00 00 76 00 00 00 28 00'
 '00 00 14 00 00 00 14 00 00 00 01 00 04 00 00 00'
 '00 00 F0 00 00 00 00 00 00 00 00 00 00 00 00 00'
 '00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 80'
 '00 00 00 80 80 00 80 00 00 00 80 00 80 00 80 80'
 '00 00 C0 C0 C0 00 80 80 80 00 00 00 FF 00 00 FF'
 '00 00 00 FF FF 00 FF 00 00 00 FF 00 FF 00 FF FF'
 '00 00 FF FF FF 00 BB BB BB BB BB BB BB BB BB BB'
 '00 00 BB BB BB BB BB BB BB BB BB BB 00 00 BB BB'
 'BB BB BB BB BB BB BB BB 00 00 BB BB BB B8 88 88'
 'BB BB BB BB 00 00 BB BB BB B8 88 88 BB BB BB BB'
 '00 00 BB BB BB B8 00 88 BB BB BB BB 00 00 BB BB'
 'BB B8 00 BB BB BB BB BB 00 00 BB BB BB BB BB BB'
 'BB BB BB BB 00 00 BB BB BB B8 00 88 BB BB BB BB'
 '00 00 BB BB BB BB 80 08 BB BB BB BB 00 00 BB BB'
 'B8 8B B8 00 8B BB BB BB 00 00 BB BB B8 00 BB 00'
 '8B BB BB BB 00 00 BB BB B8 00 BB 00 8B BB BB BB'
 '00 00 BB BB BB 80 00 00 8B BB BB BB 00 00 BB BB'
 'BB B8 00 00 8B BB BB BB 00 00 BB BB BB B8 88 88'
 'BB BB BB BB 00 00 BB BB BB BB BB BB BB BB BB BB'
 '00 00 BB BB BB BB BB BB BB BB BB BB 00 00 BB BB'
 'BB BB BB BB BB BB BB BB 00 00 BB BB BB BB BB BB'
 'BB BB BB BB 00 00'
}

CM_INPUTFILESITEM1 BITMAP "pointfile.bmp"
/*{
 '42 4D 96 01 00 00 00 00 00 00 76 00 00 00 28 00'
 '00 00 18 00 00 00 18 00 00 00 01 00 04 00 00 00'
 '00 00 20 01 00 00 00 00 00 00 00 00 00 00 00 00'
 '00 00 10 00 00 00 00 00 00 00 00 00 80 00 00 80'
 '00 00 00 80 80 00 80 00 00 00 80 00 80 00 80 80'
 '00 00 C0 C0 C0 00 80 80 80 00 00 00 FF 00 00 FF'
 '00 00 00 FF FF 00 FF 00 00 00 FF 00 FF 00 FF FF'
 '00 00 FF FF FF 00 FF FF FF FF FF FF FF FF FF FF'
```

```
 'FF FF FF 00 00 00 00 00 00 00 00 08 8F FF 0A'
 'AA AA AA AA AA AA AA AA 08 8F FF 0A AA AA AA AA'
 'AA AA AA AA 08 8F FF 0A 99 9A AA AA AA AA AA AA'
 '08 8F FF 0A AA AA AA AA AA AA AA AA 08 8F FF 0A'
 'AA AA AA AA AA AA AA AA 08 8F FF 00 00 00 00 00'
 '00 00 00 00 08 8F FF 88 88 88 88 88 88 88 88 88'
 '88 8F FF FF FF FF FF FF FF FF FF FF FF FF FF FF'
 'FF FF FF FF FF FF FF FF FF FF FF F0 00 00 00 00'
 '00 00 0F FF FF FF FF F0 FF FF FF FF FF FF 0F FF'
 'FF FF FF F0 F0 00 FF FF FF FF 0F FF FF FF FF F0'
 'FF FF FF F0 00 0F 0F FF FF FF FF F0 FF FF FF FF'
 'FF FF 0F FF FF FF FF F0 FF FF FF 00 0F FF 0F FF'
 'FF FF FF F0 FF FF FF FF FF F0 8F FF FF FF FF F0'
 'F0 00 FF FF FF 08 8F FF FF FF FF F0 FF FF FF FF'
 '00 88 8F FF FF FF FF F0 FF FF FF F0 88 88 8F FF'
 'FF FF FF F0 00 00 00 08 88 88 8F FF FF FF FF FF'
 'FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF'
 'FF FF FF FF FF FF'
}
*/

CM_POPUPITEM1 BITMAP "tinfile.bmp"
/*{
 '42 4D 96 01 00 00 00 00 00 00 76 00 00 00 28 00'
 '00 00 18 00 00 00 18 00 00 00 01 00 04 00 00 00'
 '00 00 20 01 00 00 00 00 00 00 00 00 00 00 00 00'
 '00 00 10 00 00 00 00 00 00 00 00 00 80 00 00 80'
 '00 00 00 80 80 00 80 00 00 00 80 00 80 00 80 80'
 '00 00 C0 C0 C0 00 80 80 80 00 00 00 FF 00 00 FF'
 '00 00 00 FF FF 00 FF 00 00 00 FF 00 FF 00 FF FF'
 '00 00 FF FF FF 00 FF FF FF FF FF FF FF FF FF FF'
 'FF FF FF 00 00 00 00 00 00 00 00 00 08 8F FF 0A'
 'AA AA AA AA AA AA AA AA 08 8F FF 0A AA AA AA AA'
 'AA AA AA AA 08 8F FF 0A A9 99 9A AA AA AA AA AA'
 '08 8F FF 0A AA AA AA AA AA AA AA AA 08 8F FF 0A'
 'AA AA AA AA AA AA AA AA 08 8F FF 00 00 00 00 00'
 '00 00 00 00 08 8F FF 88 88 88 88 88 88 88 88 88'
 '88 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF'
 'FF FF FF FF FF FF FF FF FF FF FF FF 00 00 00 00'
 '00 00 00 FF FF FF FF FF 0F FF FF FF FF FF F0 FF'
 'FF FF FF FF 0F FF FF FF 00 FF F0 FF FF FF FF FF'
 '0F FF FF FF FF FF F0 FF FF FF FF FF 0F F0 00 0F'
 'FF FF F0 FF FF FF FF FF 0F FF FF FF FF FF F0 FF'
 'FF FF FF FF 0F FF FF F0 0F FF F0 FF FF FF FF FF'
 '0F FF FF FF FF FF 08 FF FF FF FF FF 0F F0 00 FF'
 'FF F0 88 FF FF FF FF FF 0F FF FF FF F0 08 88 FF'
 'FF FF FF FF 0F FF FF FF 08 88 88 FF FF FF FF FF'
 '00 00 00 00 88 88 88 FF FF FF FF FF FF FF FF FF'
 'FF FF FF FF FF FF'
}

*/
```

TIN Data Restructuring Code

The following gives the complete codes for the data restructuring, that is from the normal TIN structure to triangle's sides structure. This is for the purpose of TIN-based applications such as contouring.

```
//*************************************************************//
//       A program to restruct TIN structure to SID structure    //
//         Input : TIN file, NBR file                            //
//         Output : Sides file                                   //
//                                                               //
//*************************************************************//

#include <iostream.h>
#include <fstream.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <string>

#define maxtriangle 15000
#define maxside 20000


class Restruct
  {
   public:
      // Triangle table structure
      struct ttri
        {
         int Node1;
         int Node2;
         int Node3;
        };
      ttri* Tri3Nodes[maxtriangle];

      // Triangle neighbour structure
      struct tnbr
        {
         int trino;
         int numnbr;
         int Nbr1, Nbr2, Nbr3;
        };
      tnbr* TriNbr[maxtriangle];

      // Triangle sides structure
      struct tsid
        {
         int Node1;
         int Node2;
```

```
        int RightTri;
        int LeftTri;
       };
     tsid* TriSides[maxside];

    // structure for 3 sides table
    struct trs
      {
       int Side[3];
      };
    trs* Tri3Sides[maxtriangle];

    int ntri;
    int nsid;
    int idx[maxside];
    int ridx[maxside];

    FILE* TINfile;
    FILE* NBRfile;

    Restruct();   // constructor
    ~Restruct();  // destructor

    void PressAnyKey();
    void ReadTINFile();
    void ReadNBRFile();
    void WriteOutputFile();
    bool ExistingSides(int, int, int&);
    void DoSide(int, int, int, int);
    void MakeSides();
    bool Less(int, int);
    void Swap(int, int);
    void QuickSort(int, int);
    void Sort();
  };


// some definitions
Restruct :: Restruct()         // constructor
  {
   // Allocate memory for TIN data
   for (int t = 0; t < maxtriangle; t ++)
     {
      Tri3Nodes[t] = new ttri;
     }

   // Allocate memory for TIN neighbour data
   for (int n = 0; n < maxtriangle; n ++)
     {
      TriNbr[n] = new tnbr;
     }

   // Allocate memory for Triangle sides data
   for (int s = 0; s < maxtriangle; s ++)
     {
      TriSides[s] = new tsid;
     }

   for (int k = 0; k < maxtriangle; k ++)
     {
```

```
      Tri3Sides[k] = new trs;
     }
  }

Restruct :: ~Restruct()        // destructor
  {
   // Deallocate memory for TIN data
   for (int r = 0; r < maxtriangle; r ++)
     {
      delete [] Tri3Nodes[r];
     }

   // Allocate memory for TIN neighbour data
   for (int n = 0; n < maxtriangle; n ++)
     {
      delete [] TriNbr[n];
     }

   // Allocate memory for Triangle sides data
   for (int s = 0; s < maxtriangle; s ++)
     {
      delete [] TriSides[s];
     }

   for (int k = 0; k < maxtriangle; k ++)
     {
      delete [] Tri3Sides[k];
     }
  }


void Restruct :: PressAnyKey()
  {
   cout << endl;
   cout << "Press any key to continue" << endl;
   getch();
   cout << endl;
  }

void Restruct :: ReadTINFile()
  {
   string charNode1, charNode2, charNode3;
   ifstream TINfile("c:\\data\\sercomt$.tin");

   if (! TINfile)
     cout << "Unable to open TIN file" << endl;
   else
     {
      cout << "TIN file is opened and okay for reading." << endl;
      cout << endl;

      // reads the file header
      TINfile >> charNode1 >> charNode2 >> charNode3;
      ntri = 0;
      do
        {
         //ntri ++;
         TINfile >> Tri3Nodes[ntri]->Node1
                 >> Tri3Nodes[ntri]->Node2
                 >> Tri3Nodes[ntri]->Node3;
         ntri ++;
```

```
        } while (! TINfile.eof());
      }

   cout << endl;
   cout << "ntri: " << ntri << endl;
   TINfile.close();
  }

void Restruct :: ReadNBRFile()
  {
   string chartrino;
   string charnumnbr;
   string charNbr1;
   string charNbr2;
   string charNbr3;

   ifstream NBRfile("c:\\data\\sercomt$.nbr");

   if (! NBRfile)
     cout << "Unable to open file" << endl;
   else
     {
      cout << "TIN NBR file is opened and okay for reading." << endl;
      cout << endl;

      NBRfile >> chartrino >> charnumnbr
              >> charNbr1 >> charNbr2 >> charNbr3;

      ntri = 0;
      do
        {
         ntri ++;
         NBRfile >> TriNbr[ntri]->trino
                 >> TriNbr[ntri]->numnbr
                 >> TriNbr[ntri]->Nbr1
                 >> TriNbr[ntri]->Nbr2
                 >> TriNbr[ntri]->Nbr3;

        } while (! NBRfile.eof());
     }

   NBRfile.close();
  }

void Restruct :: WriteOutputFile()
  {
   char* chartrino = "Tri#";
   char* charSide1 = "Side1";
   char* charSide2 = "Side2";
   char* charSide3 = "Side3";

   char* charsideno = "Side#";
   char* charNode1 = "Node1";
   char* charNode2 = "Node2";
   char* charRightTri = "RightTri";
   char* charLeftTri = "LeftTri";

   // write the tri sides output file
   char TRISIDESfilename [] = "c:\\data\\sercomt$.sid";
   FILE* TRISIDESfile = fopen(TRISIDESfilename, "w");
```

```
     fprintf(TRISIDESfile, "%5s %5s %5s %5s \n", chartrino, charSide1,
charSide2, charSide3);

   for (int i = 1; i <= ntri-1; i ++)
     {
      fprintf(TRISIDESfile, "%5d %5d %5d %5d \n", i, Tri3Sides[i]->Side[0],
                                                     Tri3Sides[i]->Side[1],
                                                    Tri3Sides[i]->Side[2]);
     }
   fclose(TRISIDESfile);

   // write the sides output file
   char SIDESfilename [] = "c:\\data\\sercomt$.trs";
   FILE* SIDESfile = fopen(SIDESfilename, "w");

    fprintf(SIDESfile, "%5s %5s %5s %5s %5s \n", charsideno, charNode1,
charNode2,
                                                 charRightTri, charLeftTri);

   for (int i = 1; i <= nsid; i ++)
     {
      fprintf(SIDESfile, "%5d %5d %5d %7d %8d \n", i, TriSides[i]->Node1,
                                                      TriSides[i]->Node2,
                                                     TriSides[i]->RightTri,
                                                     TriSides[i]->LeftTri);
     }
   fclose(SIDESfile);

  }


bool Restruct :: ExistingSides(int n1, int n2, int& s)
  {
   bool found;

   s = 0;
   found = false;

   do
     {
      found = (n1 == TriSides[s]->Node1) && (n2 == TriSides[s]->Node2);
      if (! found)
        s ++;
     } while ((! found) && (s <= nsid));

   return found;
  }

void Restruct :: DoSide(int t, int n1, int n2, int snbr)
  {
   int h;
   int s;

   if (n1 > n2)
     {
      h = n1;
      n1 = n2;
      n2 = h;
     }

   if (ExistingSides(n1, n2, s))
```

```
     {
      TriSides[s]->RightTri = t;
      Tri3Sides[t]->Side[snbr] = s;
     }

   else
     {
      nsid ++;
      TriSides[nsid] = new tsid;
      TriSides[nsid]->Node1 = n1;
      TriSides[nsid]->Node2 = n2;
      TriSides[nsid]->LeftTri = t;
      TriSides[nsid]->RightTri = 0;

      Tri3Sides[t]->Side[snbr] = nsid;
     }
  }


void Restruct :: MakeSides()
  {
   ifstream TINfile("c:\\data\\sercomt$.tin");
   nsid = 0;

   for (int t = 1; t < ntri; t ++)
     {
             TINfile  >>  Tri3Nodes[t]->Node1  >>  Tri3Nodes[t]->Node2  >>
Tri3Nodes[t]->Node3;
      printf("\r");
      cout << "t: " << t;
      DoSide(t, (int)Tri3Nodes[t]->Node1, (int)Tri3Nodes[t]->Node2, 0);
      DoSide(t, (int)Tri3Nodes[t]->Node2, (int)Tri3Nodes[t]->Node3, 1);
      DoSide(t, (int)Tri3Nodes[t]->Node3, (int)Tri3Nodes[t]->Node1, 2);
     }

   TINfile.close();
  }

bool Restruct :: Less(int i, int j)
  {
   int ni1;
   int ni2;
   int nj1;
   int nj2;

   ni1 = (int)TriSides[idx[i]]->Node1;
   ni2 = (int)TriSides[idx[i]]->Node2;
   nj1 = (int)TriSides[idx[j]]->Node1;
   nj2 = (int)TriSides[idx[j]]->Node2;

   return (ni1 < nj1) || ((ni1 == nj1) && (ni2 < nj2));
  }

void Restruct :: Swap(int i, int j)
  {
   int h;
   h = idx[i];
   idx[j] = h;
  }

void Restruct :: QuickSort(int l, int r)
```

```
  {
   int m;
   int i;
   int j;

   i = l;
   j = r;
   m = (l + r) / 2;

   do
     {
      do
        {
         i ++;
        } while (Less(i, m));

      do
        {
         j --;
        } while (Less(m, j));

      if (i <= j)
        {
         if (i == m)
           m = j;
         else
           if (j == m)
             m = i;
         Swap(i, j);
         i ++;
         j ++;
        }

     } while (i > j);

   if (l < j)
      QuickSort(l, j);
   if (i < r)
      QuickSort(i, r);
  }

void Restruct :: Sort()
  {
   int i;
   int j;

   for (i = 0; i < nsid; i ++)
     {
      idx[i] = i;
     }

   QuickSort(1, nsid);
   for (i = 0; i < nsid; i ++)
     {
      ridx[idx[i]] = i;
     }

   for (i = 0; i < ntri; i ++)
     {
      for (j = 0; j <= 2; j ++)
```

```
        Tri3Sides[i]->Side[j] = ridx[Tri3Sides[i]->Side[j]];
      }
  }

// main program
void main()
  {
   Restruct Re;
   Re.ReadTINFile();
   Re.ReadNBRFile();
   cout << "Processing ...." << endl;
   Re.MakeSides();
   cout << endl;
   //cout << "Sort ..." << endl;
   //Re.Sort();
   cout << endl;
   cout << "Output ..." << endl;
   Re.WriteOutputFile();
  }
```

TIN Topology Code

The following gives the complete codes for the TIN neighbouring program.

```
//************************************************//
//       A program to generate TIN topology       //
//         Input : TIN file                       //
//         Output : TIN neighbours (.NBR) file    //
//                                                //
//************************************************//

#include <iostream.h>
#include <fstream.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <string>

#define maxtriangle 15000

int evencheck(int value)
  {
   return (value & 1) == 0;
  }

class Neighbour
  {
   public:
     int TriNum;
     int TotalNeighbour;
     int Nbr[3];
     Neighbour()
       {
        TriNum = 0;
        TotalNeighbour = 0;
        Nbr[0] = Nbr[1] = Nbr[2] = 0;
       };
     ~Neighbour(){};
  };


class TinNeighbour
  {
   public:
      // structure for the TINs
      struct ThreeNodes
        {
         int Node[3];
        };
      ThreeNodes* tri[maxtriangle];

     // some global variables
```

```
    FILE* TINfile;
    FILE* NBRfile;
    int t;
    int ntri;
    bool isTriangleNode;

    // some operations
    TinNeighbour();
    ~TinNeighbour();

    void GetTINfile();
    void GetNeighbourfile();
    bool Less(int a, int b);
    bool Greater(int, int);
    void Swap(int&, int&);
    void NodeOrder(int&, int&, int&);
    void AddNbrtoFile(Neighbour);
    void MakeTinNeighbour();
    void PressAnyKey();
  };

TinNeighbour :: TinNeighbour()
  {
   // Allocate memory for TINs
   for (int t = 0; t < maxtriangle; t ++)
     {
      tri[t] = new ThreeNodes;
     }
  }

TinNeighbour :: ~TinNeighbour()
  {
   // Deallocate memory for TIN neighbour
   for (int t = maxtriangle-1; t >= 0; t --)
     {
      delete [] tri[t];
     }
  }

void TinNeighbour :: GetTINfile()
  {
   int t;
   string charNode1, charNode2, charNode3;

   // Reads the TIN file
   //ifstream TINfile = "c:\\data\\sercomt$.tin";
   ifstream TINfile = "c:\\data\\lakebo.tin";

   if (TINfile == NULL)
     {
      cerr << "Could not open TIN file !" << endl;
      exit(1);
     }
   else
     {
      cout << endl;
      cout << "The TIN file is opened" << endl;

      // reads the file header
      TINfile >> charNode1 >> charNode2 >> charNode3;
```

```
      t = 0;
      do
        {
         printf("\r");
         t++;
         cout << "reads Triangle  ... " << t;

         for (int n = 0; n < 3; n ++)
           {
            TINfile >> tri[t] -> Node[n];
           }

        } while (! TINfile.eof());
     }
    ntri = t-1;
    TINfile.close();
  }

void TinNeighbour :: GetNeighbourfile()
  {
   //char Neighbourfilename [] = "c:\\data\\sercomt$.nbr";
   //char Neighbourfilename [] = "c:\\data\\lakebo.nbr";

   NBRfile = fopen(Neighbourfilename, "w");

   char* TINnum = "Tri#";
   char* NumofNbr = "NumofNeighbours";
   char* Nbr1 = "Nbr1";
   char* Nbr2 = "Nbr2";
   char* Nbr3 = "Nbr3";

   fprintf(NBRfile, "%8s %8s %8s %8s %8s \n", TINnum, NumofNbr, Nbr1, Nbr2,
Nbr3);
   if (NBRfile == NULL)
     {
      cerr << "Could not open TIN Neighbour file !" << endl;
      exit(1);
     }
   else
     {
      cout << endl;
      cout << "preparing TIN Neighbour file ... " << endl;
      cout << endl;
     }
  }

// Function: Less
bool TinNeighbour :: Less(int a, int b)
  {
   return (a < b);
  }

// Function: Greater
bool TinNeighbour :: Greater(int a, int b)
  {
   return (a > b);
  }

//Procedure: Swap
void TinNeighbour :: Swap(int& a, int& b)
  {
```

```
   int temp;

   temp = a;
   a = b;
   b = temp;
  }

// Procedure: NodeOrder
void TinNeighbour :: NodeOrder(int& a, int& b, int& c)
  {
   if (Greater(a, b))
       Swap(a, b);
   if (Greater(b, c))
       Swap(b, c);
   if (Greater(a, b))
       Swap(a, b);
  }

void TinNeighbour :: AddNbrtoFile(Neighbour Element)
  {
   fprintf(NBRfile, "%7d %8d %14d %8d %8d \n", Element.TriNum,
                                               Element.TotalNeighbour,
                                               Element.Nbr[0],
                                               Element.Nbr[1],
                                               Element.Nbr[2]);
  }

void TinNeighbour :: MakeTinNeighbour()
  {
   int CommonNode;
   int NumofNbr;
   bool isTriangleNode;

   int tt;
   int i;

   for (t = 1; t <= ntri; t ++)
      {
       printf("\r");
       cout << "Making TIN neighbours ... " << t;

       Neighbour Element;
       NumofNbr = -1;

       for (tt = 1; (tt <= ntri) && (NumofNbr <= 3); tt ++)
         {
          if (t == tt) continue;

          CommonNode = 0;

          for (i = 0; (i < 3) && (CommonNode <= 2); i ++)
              {
               isTriangleNode = (tri[t] -> Node[i] == tri[tt] -> Node[0]) ||
                                (tri[t] -> Node[i] == tri[tt] -> Node[1]) ||
                                (tri[t] -> Node[i] == tri[tt] -> Node[2]);

               if (isTriangleNode == true)
                 {
                  CommonNode ++;
                  if (CommonNode == 2)
                    {
```

```
                    NumofNbr ++;
                    Element.Nbr[NumofNbr] = tt;
                    Element.TotalNeighbour = NumofNbr + 1;
                   }
                 }
               }
          }

       Element.TriNum = t;
       AddNbrtoFile(Element);
      }

   fclose(NBRfile);
  }

void TinNeighbour :: PressAnyKey()
  {
   cout << endl;
   cout << "Press any key to continue" << endl;
   getch();
   cout << endl;
  }

// Main program
void main()
  {
   TinNeighbour TINNBR;
   TINNBR.GetTINfile();
   TINNBR.GetNeighbourfile();
   TINNBR.MakeTinNeighbour();
   TINNBR.PressAnyKey();
  }
```

Contouring Code

The following gives the complete code for the contouring from TIN data structures.

```
//*********************************************************//
//                                                         //
//       A Program to interpolate contours from TIN        //
//           Input: point (XYZ) file, TRS and SID files    //
//           Output: Contour interpolation file (.CON)     //
//                                                         //
//*********************************************************//

#include <iostream.h>
#include <stdlib.h>
#include <stdio.h>
#include <fstream.h>
#include <conio.h>
#include <math.h>
#include <string>

#define maxpoint 7000
#define maxtriangle 10000
#define maxside 16000
#define accy 0.001

class Contour
  {
   public:
      struct PointStruct
        {
         float x;
         float y;
         float z;
        };
      PointStruct* Point[maxpoint];

      struct TriSideStruct
        {
         int Node1;
         int Node2;
         int RightTri;
         int LeftTri;
        };
      TriSideStruct* TriSides[maxside];

      struct Tri3SidesStruct
        {
         int Side[3];
        };
      Tri3SidesStruct* Tri3Sides[maxtriangle];

      bool utri[maxtriangle];
```

```
      string charX;
      string charY;
      string charZ;
      int npnt, ntri, nside;
      int Side1, Side2, Side3;
      float x, y, z, xNd1, yNd1, xNd2, yNd2;
      float xmin, xmax, ymin, ymax;
      int interval;
      int SegNr, Hreq, prevH;
      float Hmin, Hmax, rangeH;
      float zNd1, zNd2;
      int ii;
      int t, s, i, j, k;
      FILE* interpolfile;

      void PressAnyKey();
      void PrintInput();
      void ReadInputFiles();
      void OpenOutputFile();
      void AddToFile(float, float);
      void GetMaxMinHeights();
      void GetCoordinate(int, float&, float&,
                              float&, float&,
                              float&, float&);
      void Interpolate(int, int, int);
      bool CheckSide(int);
      int FindFirstTri(int&, int&);
      void FindOtherSide(int, int, int&);
      void FindNextTri(int, int, int&);
      void GetContours(int);
      void GetInterval(int&);
      void MakeContouring();

      Contour();    // constructor
      ~Contour();   // deconstructor
  };

// Implementation of the Contour class
Contour :: Contour()
  {
   // Allocate memory for Points
   for (int i = 0; i < maxpoint; i ++)
      {
       Point[i] = new PointStruct;
      }

   // Allocate memory for Triangle sides
   for (int j = 0; j < maxside; j ++)
      {
       TriSides[j] = new TriSideStruct;
      }

   // Allocate memory for Trinagle 3 sides
   for (int k = 0; k < maxtriangle; k ++)
      {
       Tri3Sides[k] = new Tri3SidesStruct;
      }
  }

Contour :: ~Contour()
```

```
  {
   // Deallocate memory for points
   for (int i = 0; i < maxpoint; i ++)
       {
        delete [] Point[i];
       }

  // Deallocate memory for Triangles sides
  for (int j = 0; j < maxside; j ++)
       {
        delete [] TriSides[j];
       }

 // Deallocate memory for Triangle 3 sides
 for (int k = 0; k < maxtriangle; k ++)
       {
        delete [] Tri3Sides[k];
       }
  }


void Contour :: PressAnyKey()
  {
   cout << endl;
   cout << endl;
   cout << "Press any key to continue" << endl;
   getch();
   cout << endl;
  }

void Contour :: ReadInputFiles()  // Point, Sides, and Triangle sides files
  {
   // Reads the Point file
   ifstream pointfile("c:\\data\\sercomb.xyz");
     if (! pointfile)
       {
        cout << "Could not open point file !" << endl;
        exit(1);
       }
       else
         {
          cout << endl;
          cout << "The point file is opened" << endl;

          pointfile >> charX >> charY >> charZ;

          npnt = 0;
          do
            {
             npnt ++;
             printf("\r");
             cout << "Reading the points ... " << npnt;
            pointfile >> Point[npnt]->x >> Point[npnt]->y >> Point[npnt]->z;
             //npnt ++;
            } while (! pointfile.eof());
          npnt = npnt - 1;
         }

   // to print the node coordinates
   //cout << endl;
   //cout << "The nodes coordinates:" << endl;
```

```
//cout << Point[1]->x << " " << Point[1]->y <<" " << Point[1]->z << endl;
//cout << Point[2]->x << " " << Point[2]->y <<" " << Point[2]->z << endl;
//cout << Point[3]->x << " " << Point[3]->y <<" " << Point[3]->z << endl;
//cout << endl;
//PressAnyKey();

// Reads the Triangle sides file
string charSideNo;
string charNode1;
string charNode2;
string charRightTri;
string charLeftTri;

ifstream trisidfile("c:\\data\\sercomt$.trs");

   if (! trisidfile)
    {
     cout << "Could not open triangle sides file !" << endl;
     exit(1);
    }
    else
     {
      cout << endl;
      cout << "The triangle sides file is opened" << endl;

      trisidfile >> charSideNo
                 >> charNode1
                 >> charNode2
                 >> charRightTri
                 >> charLeftTri;

      nside = 0;
      do
        {
         nside ++;
         printf("\r");
         cout << "Reading the triangle sides ... " << nside;
         trisidfile >> nside
                    >> TriSides[nside]->Node1
                    >> TriSides[nside]->Node2
                    >> TriSides[nside]->RightTri
                    >> TriSides[nside]->LeftTri;
         //nside ++;
        } while (! trisidfile.eof());
      nside = nside - 1;
     }

// to print the triangle sides data
cout << endl;
cout << endl;

ifstream tri3sidfile("c:\\data\\sercomt$.sid");
  string charTriNo;
  string charSide1;
  string charSide2;
  string charSide3;

   if (! tri3sidfile)
    {
     cout << "Could not open triangles file !" << endl;
     exit(1);
```

```
       }
       else
        {
         cout << endl;
         cout << "The triangles file is opened" << endl;

         tri3sidfile >> charTriNo
                     >> charSide1
                     >> charSide2
                     >> charSide3;

         ntri = 0;
         do
           {
            ntri ++;
            printf("\r");
            cout << "Reading the triangles ... " << ntri;
            tri3sidfile >> ntri
                        >> Tri3Sides[ntri]->Side[0]
                        >> Tri3Sides[ntri]->Side[1]
                        >> Tri3Sides[ntri]->Side[2];
            //ntri ++;
           } while (! tri3sidfile.eof());
         ntri = ntri - 1;
        }

   // to print the three sides
   cout << endl;
   trisidfile.close();
   tri3sidfile.close();
  }

void Contour :: OpenOutputFile()
  {
   char interpolfilename [] = "c:\\data\\sercom10.con";
   interpolfile = fopen(interpolfilename, "w");

   char* charx = "X";
   char* chary = "Y";
   char* charHreq = "Height";
   char* charSegNr = "SegNumber";

    fprintf(interpolfile, "%3s %8s %8s %8s \n", charx, chary, charHreq,
charSegNr);
   if (interpolfile == NULL)
     {
      cerr << "Could not open the interpolation file !" << endl;
      exit(1);
     }
   else
     {
      cout << endl;
      cout << "The file is ready for output" << endl;
     }

  }

void Contour :: AddToFile(float x, float y)
  {
   fprintf(interpolfile, "%3.1f %8.1f %5d %5d \n", x, y, Hreq, SegNr);
  }
```

```
void Contour :: GetMaxMinHeights()
  {
   float x, y, z;

   // Reads the Point file
   ifstream pointfile("c:\\data\\sercomb.xyz");
     if (! pointfile)
       {
        cerr << "Could not open point file !" << endl;
        exit(1);
       }
       else
        {
         cout << endl;
         cout << "The point file is opened" << endl;

         pointfile >> charX >> charY >> charZ;

         npnt = 0;
         pointfile >> x >> y >> z;

         xmin = x;
         xmax = x;

         ymin = y;
         ymax = y;

         Hmax = z;
         Hmin = z;

         do
           {
            npnt ++;
            pointfile >> x >> y >> z;
            if (xmax < x)
                xmax = x;

            if (xmin > x)
                xmin = x;

            if (ymax < y)
                ymax = y;

            if (ymin > y)
                ymin = y;

            if (Hmax < z)
                Hmax = z;
            if (Hmin > z)
                Hmin = z;

            //npnt ++;
           } while (! pointfile.eof());

         rangeH = Hmax - Hmin;
         cout << endl;
         cout << "Max height  : " << Hmax << endl;
         cout << "Min height  : " << Hmin << endl;
         cout << "The H range : " << rangeH << endl;
        }
```

```
   pointfile.close();
  }

void Contour :: GetCoordinate(int s, float& xNd1, float& yNd1,
                                     float& xNd2, float& yNd2,
                                     float& zNd1, float& zNd2)
  {
   int Nd1, Nd2;
   Nd1 = (int)TriSides[s]->Node1;
   Nd2 = (int)TriSides[s]->Node2;

   xNd1 = Point[Nd1]->x;
   yNd1 = Point[Nd1]->y;
   zNd1 = Point[Nd1]->z;

   xNd2 = Point[Nd2]->x;
   yNd2 = Point[Nd2]->y;
   zNd2 = Point[Nd2]->z;
  }

void Contour :: Interpolate(int s, int SegNr, int Hreq)
  {
   float dh, dhi;
   float dx, dy, x, y;

   GetCoordinate(s, xNd1, yNd1, xNd2, yNd2, zNd1, zNd2);

   dh = zNd1 - zNd2;
   if (abs(dh) < accy)
     dh = dh + accy;
   else
     {
      dx = xNd1 - xNd2;
      dy = yNd1 - yNd2;
      dhi = Hreq - zNd1;
      x = xNd1 + ((dx * dhi) / dh);
      y = yNd1 + ((dy * dhi) / dh);

      // to check the x range, dont write the out of the limit x, y
      if ( !(x < xmin) )
       {
        AddToFile(x, y);
        //cout << endl;
        //cout << "side: " << s << endl;
        //cout << "interpolated x, y, Hreq, SegNr: " << endl;
        //cout << x << " " << y << " " << Hreq << " " << SegNr << endl;
        //cout << endl;
       }
     }
  }

bool Contour :: CheckSide(int s)
  {
   int n1, n2;
   float z1, z2;

   n1 = TriSides[s]->Node1;
   n2 = TriSides[s]->Node2;
   z1 = Point[n1]->z;
   z2 = Point[n2]->z;
   z1 = z1 + 0.001;
```

```
   z2 = z2 + 0.001;
   return ((z1 < Hreq) && (z2 > Hreq)) || ((z2 < Hreq) && (z1 > Hreq));
  }

int Contour :: FindFirstTri(int& t, int& s)
  {
   int i;
   bool found;

   //t = 0;
   t = 1;
   found = false;
   while ((t < ntri) && (! found))
     {
      if (! utri[t])
        {
         i = 0;
         //i = 1;
         while ((i < 3) && (! found))
           {
            s = Tri3Sides[t]->Side[i];
            if (CheckSide(s))
               found = true;
            else
               i++;
           }
        }
      if (! found)
         t ++;
     }

   if (! found)
      //t = -1;
      t = 0;

   return t;
  }

void Contour :: FindOtherSide(int t, int s, int& nexts)
  {
   int i;
   bool found;

   found = false;
   i = 0;
   //i = 1;
   while ((i < 3) && (! found))
     {
      nexts = Tri3Sides[t]->Side[i];
      if ((s != nexts) && CheckSide(nexts))
        found = true;
      else
        i++;
     }
  }

void Contour :: FindNextTri(int t, int s, int& nextt)
  {
   if (t == (int)TriSides[s]->RightTri)
     nextt = (int)TriSides[s]->LeftTri;
```

```
   else
     nextt = (int)TriSides[s]->RightTri;
  }

void Contour :: GetContours(int Hreq)
  {
   bool done;
   bool secondpart;
   int startt, starts;
   int s, t;
   float x, y;
   int nextt, nexts;

   //for (int i = 0; i < ntri; i ++)
   for (int i = 1; i < ntri; i ++)
     {
      utri[i] = false;
     }

      do
        {
         //while (FindFirstTri(t, s) != -1)
         while (FindFirstTri(t, s) != 0)
           {
            ii ++;
            SegNr = ii;
            startt = t;
            starts = s;
            done = false;
            secondpart = false;

            do
              {
               utri[t] = true;
               Interpolate(s, SegNr, Hreq);
               FindOtherSide(t, s, nexts);
               FindNextTri(t, nexts, nextt);
               if (nextt == startt)
                 {
                  // found closed contours
                  Interpolate(nexts, SegNr, Hreq);
                  done = true;
                 }
               //else if (nextt == -1)
               else if (nextt == 0)
                      {
                       // hit border
                       Interpolate(nexts, SegNr, Hreq);
                       if (secondpart)
                         done = true;
                       else
                         {
                          FindNextTri(startt, starts, t);
                          //if (t == -1)
                          if (t == 0)
                             done = true;
                          else
                            {
                             s = starts;
                             secondpart = true;
```

```
                            }
                          }
                        }
                  else
                   {
                    t = nextt;
                    s = nexts;
                   }
                } while (! done);

            }

          //} while ( ! ((t == -1) || (s == -1) || (z != prevH)) ) ;
          } while ( ! ((t == 0) || (s == 0) || (z != prevH)) ) ;
  }

void Contour :: GetInterval(int& interval)
  {
   cout << endl;
   cout << "Enter the required contour interval :" << endl;
   cin >> interval;
  }

void Contour :: MakeContouring()
  {
   prevH = -9999;
   Hmin = Hmin - 0.001;
   Hmax = Hmax + 0.001;
   cout << "No. of triangles :" << ntri << endl;
   cout << "No. of triangle sides :" << nside << endl;
   cout << "No. of nodes :" << npnt << endl;
   cout << endl;
   GetInterval(interval);
   if (interval > rangeH)
     {
      cout << "Sorry, no contours available at the request interval" << endl;
      exit(1);
     }
   else
     {
      Hreq = floor((Hmin / interval) + 1) * interval;
      //cout << "Interpolating the contours ..." << endl;
      //cout << endl;

      while (Hreq < Hmax)
        {
         prevH = Hreq;
         //cout << "Hreq: " << Hreq << endl;
         GetContours(Hreq);
         printf("\r");
         cout << "Interpolating contour height ..." << Hreq;
         Hreq = Hreq + interval;
        }
     }
     cout << endl;
     cout << endl;
     cout << "Contour threading complete." << endl;
     fclose(interpolfile);
  }
```

```
// main program
void main()
  {
   Contour Cont;
   Cont.ii = 0;
   Cont.ReadInputFiles();
   Cont.OpenOutputFile();
   Cont.GetMaxMinHeights();
   Cont.MakeContouring();
   Cont.PressAnyKey();
  }
```

```
// tinwin2d.rh

#define IDI_ICON1 1
#define CM_CLEAR           1125
#define CM_FILEEXIT        1126
#define CM_ABOUT           1127
#define CM_INPUTFILESITEM1    18874 // for XYZ file
#define CM_POPUPITEM1      18873 // for TIN file
```

```
//*********************************************************//
//  A Windows program to display the contours segments     //
//     Input: SegNr file                                   //
//     Output: on screen (using Windows)                   //
//                                                         //
//*********************************************************//

#include <owl/pch.h>
#include <owl/applicat.h>
#include <owl/framewin.h>
#include <owl/decframe.h>
#include <owl/statusba.h>
#include <owl/controlb.h>
#include <owl/buttonga.h>
#include <owl/gdiobjec.h>
#include <owl/chooseco.h>
#include <owl/inputdia.h>
#include <owl/opensave.h>
#include <owl/dc.h>
#include <mem.h>

#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <stdio.h>
#include <stddef.h>

//#include "plotseg.rh"
#include "tinwin2d.rh"

#define maxtriangle 20000
#define maxpoint 20000

class TSegmentDrawApp : public TApplication
  {
   public:
      TSegmentDrawApp() : TApplication() {}  // call base class constructor
      void InitMainWindow();
  };

class TSegmentWindow : public TWindow
  {
   public:
      TSegmentWindow(TWindow* parent = 0)
         : TWindow(parent) {}

      struct PointStruct
        {
         float x;
         float y;
         float z;
        };
      PointStruct* Point[maxpoint];

      struct SegmentStruct
        {
         float x;
         float y;
         int Hreq;
         int SegNr;
```

```
      };
    SegmentStruct* Segment[maxpoint];

    struct TinStruct
      {
       int Node1;
       int Node2;
       int Node3;
      };
    TinStruct* Tin[maxtriangle];

    FILE* Segmentfile;
    FILE* TINfile;
    FILE* XYZfile;

    TOpenSaveDialog :: TData fileData;

    string charx, chary, charz;
    string charNode1, charNode2, charNode3;
    string charHreq, charSegNr;
    float x, y, z;
    int k;
    int Hreq, SegNr, prevSegNr;
    float xmin, xmax, ymin, ymax;
    float zmin, zmax;
    float xlength, ylength;
    int npnt, ntri, nsegn;
    int MaxX, MaxY;
    float scaleX, scaleY;

    void Transform(int, float&, float&);
    void ReadSegmentFile();
    void ReadXYZTINFile();
    void GetMinMax(float&, float&,
                   float&, float&,
                   float&, float&);
    void Paint(TDC& dc, bool, TRect&);
    void EvSize(uint, TSize&);
    void CmInputFilesItem1();
    void CmPopupItem1();
    void CmFileExit()
         {PostQuitMessage(0);}
    void CmClear();
    void CmAbout();
    DECLARE_RESPONSE_TABLE(TSegmentWindow);
  };

DEFINE_RESPONSE_TABLE1(TSegmentWindow, TWindow)
  EV_WM_SIZE,
  EV_COMMAND(CM_INPUTFILESITEM1, CmInputFilesItem1),
  EV_COMMAND(CM_POPUPITEM1, CmPopupItem1),
  EV_COMMAND(CM_ABOUT, CmAbout),
  EV_COMMAND(CM_FILEEXIT, CmFileExit),
  EV_COMMAND(CM_CLEAR, CmClear),
END_RESPONSE_TABLE;


void TSegmentWindow :: CmInputFilesItem1()
  {
   ReadSegmentFile();
```

```
  }

void TSegmentWindow :: CmPopupItem1()
  {
   ReadXYZTINFile();
  }

void TSegmentWindow :: ReadSegmentFile()
  {
   // Allocate memory for segment data
   for (int i = 0; i < maxpoint; i ++)
     {
      Segment[i] = new SegmentStruct;
     }

   // Reads again the the segment file
   //ifstream Segmentfile = "c:\\data\\pent.con";
   ifstream Segmentfile = "c:\\data\\lochgrd4.con";
   //ifstream Segmentfile = "c:\\data\\sercom30.con";
   //ifstream Segmentfile = "c:\\data\\sercom50.con";
   //ifstream Segmentfile = "c:\\data\\sercom10.con";

   if (! Segmentfile)
     {
      MessageBox ("Unable to open file: Contour Segment file", "FileError",
                  MB_OK | MB_ICONEXCLAMATION);
     }
   else
     {
      // reads the file header
      Segmentfile >> charx >> chary >> charHreq >> charSegNr;
      nsegn = 0;
        do
         {
          nsegn ++;
          Segmentfile >> Segment[nsegn]->x
                      >> Segment[nsegn]->y
                      >> Segment[nsegn]->Hreq
                      >> Segment[nsegn]->SegNr;
         //nsegn ++;
         } while (! Segmentfile.eof());

     //if (Segmentfile.eof() == true)
           //MessageBox ("Finished read the Segment file", "Segment file
complete",
                  //MB_OK | MB_ICONINFORMATION);
     }
   Segmentfile.close();
  }

void TSegmentWindow :: ReadXYZTINFile()
  {
   // Allocate memory for XYZ data
   for (int k = 0; k < maxpoint; k ++)
     {
      Point[k] = new PointStruct;
     }

   // Reads the XYZ file
   //ifstream XYZfile = "c:\\data\\pent.xyz";
   ifstream XYZfile = "c:\\data\\lochgrd.xyz";
```

```
//ifstream XYZfile = "c:\\data\\sercomb.xyz";


if (!XYZfile)
  {
   MessageBox("Unable to open file: XYZ file",
              "File Error", MB_OK | MB_ICONEXCLAMATION);
  }

else
  {
   // reads the file header
   XYZfile >> charx >> chary >> charz;
   npnt = 0;
   do
     {
      npnt ++;
      XYZfile >> Point[npnt]->x
              >> Point[npnt]->y
              >> Point[npnt]->z;
     } while (! XYZfile.eof());

    //if (XYZfile.eof() == true)
     //MessageBox ("Finished read the XYZ nodes", "XYZ file complete",
                //MB_OK | MB_ICONINFORMATION);
   }

XYZfile.close();


// Allocate memory for Triangle
for (int t = 0; t < maxtriangle; t ++)
  {
   Tin[t] = new TinStruct;
  }

// Reads the TIN file
//ifstream TINfile = "c:\\data\\pent.tin";
ifstream TINfile = "c:\\data\\lochgrd.tin";
//ifstream TINfile = "c:\\data\\sercomt$.tin";

if (!TINfile)
  {
   MessageBox("Unable to open file: TIN file",
              "File Error", MB_OK | MB_ICONEXCLAMATION);
  }

else
  {
   ntri = 0;
   // reads the file header
   TINfile >> charNode1 >> charNode2 >> charNode3;

   do
     {
      ntri ++;
      TINfile >> Tin[ntri]->Node1
              >> Tin[ntri]->Node2
              >> Tin[ntri]->Node3;
     } while (! TINfile.eof());
```

```
       //if (TINfile.eof() == true)
        //MessageBox ("Finished read the TIN nodes", "TIN file complete",
                    //MB_OK | MB_ICONINFORMATION);
      }

   TINfile.close();
  }

void TSegmentWindow :: Transform(int MaxY, float& x, float& y)
  {
   x = (x - xmin) * scaleX;
   y = MaxY - ((y - ymin) * scaleY);
  }

void TSegmentWindow :: CmClear()
  {
   // to clear and redraw
   Invalidate();
  }


void TSegmentWindow :: CmAbout()
  {
   MessageBox ("Contours display with menu - by Alias Abdul-Rahman, 1999",
               "About the Program", MB_OK | MB_ICONINFORMATION);
  }

void TSegmentWindow :: GetMinMax(float& xmax, float& ymax,
                                 float& xmin, float& ymin,
                                 float& xlength, float& ylength)
  {
   // Read the segment file and get the min-max of the coordinates
   //ifstream XYZfile = "c:\\data\\pent.xyz";
   ifstream XYZfile = "c:\\data\\lochgrd.xyz";
   //ifstream XYZfile = "c:\\data\\sercomb.xyz";

   if (! XYZfile)
      MessageBox ("Unable to open file: Segment file", "FileError",
                  MB_OK | MB_ICONEXCLAMATION);

   // reads the file header
   XYZfile >> charx >> chary >> charz;
   XYZfile >> x >> y >> z;
      xmin = x;
      xmax = x;

      ymin = y;
      ymax = y;

      do
        {
         XYZfile >> x >> y >> z;

         if (xmin > x)
             xmin = x;

         if (ymin > y)
             ymin = y;

         if (xmax < x)
             xmax = x;
```

```
         if (ymax < y)
             ymax = y;

         } while (! XYZfile.eof());

      xlength = xmax - xmin;
      ylength = ymax - ymin;
  }

void TSegmentWindow :: Paint(TDC& dc, bool, TRect&)
  {
   GetMinMax(xmax, ymax, xmin, ymin, xlength, ylength);

   ReadSegmentFile();     // reads the segment
   ReadXYZTINFile();      // read the TIN

   RECT rect;
   :: GetClientRect(HWindow, &rect);

   scaleX = (rect.right - rect.left) / xlength;
   scaleY = (rect.bottom - rect.top) / ylength;

   int MaxY = (rect.bottom - rect.top);

   for (k = 1; k <= npnt-1; k ++)
     {
      Point[k]->x = (Point[k]->x - xmin) * scaleX;
      Point[k]->y = (rect.bottom - rect.top) - ((Point[k]->y - ymin)
                                                * scaleY);
     }

   // to transform the interpolated points to scale
   for (k = 1; k <= nsegn-1; k ++)
     {
      Segment[k]->x = (Segment[k]->x - xmin) * scaleX;
      Segment[k]->y = (rect.bottom - rect.top) - ((Segment[k]->y - ymin)
                                                * scaleY);
     }

   // set pen style for point display
   /*TPen pen3 = TPen(RGB(255, 0, 255), 2, PS_SOLID);
   SelectObject(dc, pen3);

   for (k = 1; k <= nsegn-1; k ++)
     {
      dc.MoveTo(Segment[k]->x, Segment[k]->y);
      dc.LineTo(Segment[k]->x, Segment[k]->y);
     }
   SelectObject(dc, pen3);
   DeleteObject(pen3);
  */
   // set pen style for TINs display
   TPen pen2 = TPen(RGB(0, 0, 255), 1, PS_SOLID);
   SelectObject(dc, pen2);

   for (k = 1; k <= ntri-1; k ++)
     {
      dc.MoveTo(Point[Tin[k]->Node1]->x,
                Point[Tin[k]->Node1]->y);
      dc.LineTo(Point[Tin[k]->Node2]->x,
                Point[Tin[k]->Node2]->y);
```

```
   dc.LineTo(Point[Tin[k]->Node3]->x,
             Point[Tin[k]->Node3]->y);
   dc.LineTo(Point[Tin[k]->Node1]->x,
             Point[Tin[k]->Node1]->y);
  }
SelectObject(dc, pen2);
DeleteObject(pen2);

// set pen for Segment display
TPen pen1 = TPen(RGB(159, 0, 0), 2, PS_SOLID);

SelectObject(dc, pen1);

float startx, starty;
float nextx, nexty;
int SegNr, nextSegNr, prevSegNr;
float x, y;
int h, nexth;



//ifstream Segmentfile("c:\\data\\pent.con");
ifstream Segmentfile("c:\\data\\lochgrd4.con");
//ifstream Segmentfile("c:\\data\\sercom30.con");
//ifstream Segmentfile("c:\\data\\sercom50.con");
//ifstream Segmentfile("c:\\data\\sercom10.con");

// reads the file header
Segmentfile >> charx >> chary >> charHreq >> charSegNr;
prevSegNr = -999;
Segmentfile >> x >> y >> h >> SegNr;
Transform(MaxY, x, y);
startx = x;
starty = y;

while (! Segmentfile.eof())
   {
    Segmentfile >> nextx >> nexty >> nexth >> nextSegNr;
    Transform(MaxY, nextx, nexty);
    if (SegNr != prevSegNr)
      {
       dc.MoveTo(x, y);
       startx = x;
       starty = y;
      }
    else
      {
       if ( (x == startx) && (y == starty) )
          {
           if (SegNr == nextSegNr)
              dc.MoveTo(x, y);
           else
              dc.LineTo(x, y);
          }
       else
       dc.LineTo(x, y);
      }

    prevSegNr = SegNr;
    SegNr = nextSegNr;
```

```
       x = nextx;
       y = nexty;
       h = nexth;
      }

   //dc.LineTo(x, y);
   dc.MoveTo(x, y);

   Segmentfile.close();
   SelectObject(dc, pen1);
   DeleteObject(pen1);

   // Deallocate memory
   for (int i = 0; i < maxpoint; i ++)
     {
      delete [] Segment[i];
     }

   for (int t = 0; t < maxtriangle; t ++)
     {
      delete [] Tin[t];
     }

   for (int k = 0; k < maxpoint; k ++)
     {
      delete [] Point[k];
     }
  }

// Force the window to repaint if resize
void TSegmentWindow :: EvSize(uint, TSize&)
  {
   :: InvalidateRect(HWindow, 0, true);
  }

void TSegmentDrawApp :: InitMainWindow()
  {
   // construct the decorated MDI frame window
   TDecoratedFrame* frame = new TDecoratedFrame(0,
                      "TIN-to-Contours ver. 1.0", new TSegmentWindow);

   // construct the status bar
   TStatusBar* sb = new TStatusBar(frame, TGadget :: Recessed);

   // construct the control bar
   TControlBar* cb = new TControlBar(frame);

   cb -> Insert(*new TButtonGadget(CM_INPUTFILESITEM1, CM_INPUTFILESITEM1,
                      TButtonGadget :: Command));
   cb -> Insert(*new TButtonGadget(CM_POPUPITEM1, CM_POPUPITEM1,
                      TButtonGadget :: Command));

   cb -> Insert(*new TSeparatorGadget);

   cb -> Insert(*new TButtonGadget(CM_CLEAR, CM_CLEAR,
                      TButtonGadget :: Command));

   cb -> Insert(*new TSeparatorGadget);

   cb -> Insert(*new TButtonGadget(CM_FILEEXIT, CM_FILEEXIT,
                      TButtonGadget :: Command));
```

```
   cb -> Insert(*new TSeparatorGadget);

   cb -> Insert(*new TButtonGadget(CM_ABOUT, CM_ABOUT,
                    TButtonGadget :: Command));

   cb -> SetHintMode(TControlBar :: PressHints);

   // insert the status and control bar into the frame
   frame -> Insert(*sb, TDecoratedFrame :: Bottom);
   frame -> Insert(*cb, TDecoratedFrame :: Top);

   // set the main window and its menu
   SetMainWindow(frame);
   GetMainWindow() -> AssignMenu("COMMANDS");
   EnableCtl3d(true);
  }


int OwlMain(int /*argc*/, char* /*argv*/ [])
  {
   return TSegmentDrawApp().Run();
  }
```

```
#include "tinwin2d.rh"

COMMANDS MENU
{
 POPUP "&InputFiles"
 {
  MENUITEM "&XYZ File", CM_INPUTFILESITEM1
  MENUITEM "&TIN File", CM_POPUPITEM1
 }

 POPUP "&Redraw"
 {
  MENUITEM "&Clear and Redraw", CM_CLEAR
  MENUITEM "E&xit", CM_FILEEXIT
 }

 MENUITEM "&About", CM_ABOUT
}

STRINGTABLE
  {
   CM_INPUTFILESITEM1,  "Get the XYZ file"
   CM_POPUPITEM1,       "Get the TIN file"
   CM_CLEAR,            "Redraw the TINs"
   CM_FILEEXIT,         "Exit the program ..."
   CM_ABOUT,            "About the program ..."
  }

CM_CLEAR BITMAP
{
 '42 4D 66 01 00 00 00 00 00 00 76 00 00 00 28 00'
 '00 00 14 00 00 00 14 00 00 00 01 00 04 00 00 00'
 '00 00 F0 00 00 00 00 00 00 00 00 00 00 00 00 00'
 '00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 80'
 '00 00 00 80 80 00 80 00 00 00 80 00 80 00 80 80'
 '00 00 C0 C0 C0 00 80 80 80 00 00 00 FF 00 00 FF'
 '00 00 00 FF FF 00 FF 00 00 00 FF 00 FF 00 FF FF'
 '00 00 FF FF FF 00 91 77 77 77 77 77 77 77 77 99'
 '00 00 79 77 77 77 77 77 77 77 79 17 00 00 77 17'
 '77 87 77 77 77 79 91 77 00 00 77 77 80 08 88 87'
 '77 77 77 77 00 00 77 78 00 00 08 88 77 77 77 77'
 '00 00 77 78 0A 0A A0 08 88 87 77 77 00 00 77 78'
 '0A 0A AA A0 88 88 77 77 00 00 77 80 AA 0A AA AA'
 '00 88 77 77 00 00 77 80 AA A0 AA A0 A0 88 77 77'
 '00 00 77 0A AA A0 AA 0A A0 88 71 17 00 00 77 0A'
 'AA A0 00 AA A0 88 79 97 00 00 78 0A AA A0 AA AA'
 'A0 88 77 77 00 00 78 00 00 00 AA AA A0 88 77 77'
 '00 00 78 88 87 78 0A AA A0 88 77 77 00 00 77 77'
 '77 77 80 AA A0 88 77 77 00 00 77 71 77 77 78 0A'
 'A0 88 77 77 00 00 77 19 77 77 77 80 A0 88 77 77'
 '00 00 77 97 77 77 77 78 00 88 79 77 00 00 71 77'
 '77 77 77 77 88 87 71 19 00 00 19 77 77 77 77 77'
 '77 77 77 71 00 00'
}


CM_FILEEXIT BITMAP
{
 '42 4D 66 01 00 00 00 00 00 00 76 00 00 00 28 00'
 '00 00 14 00 00 00 14 00 00 00 01 00 04 00 00 00'
```

```
 '00 00 F0 00 00 00 00 00 00 00 00 00 00 00 00'
 '00 00 00 00 00 00 00 00 00 00 00 80 00 00 80'
 '00 00 00 80 80 00 80 00 00 00 80 00 80 00 80 80'
 '00 00 C0 C0 C0 00 80 80 80 00 00 00 FF 00 00 FF'
 '00 00 00 FF FF 00 FF 00 00 00 FF 00 FF 00 FF FF'
 '00 00 FF FF FF 00 EE EE EE EE EE EE EE EE EE EE'
 '00 00 EE EE FF EF EF FE EE EE EE EE 00 00 EE EE'
 'FF EE FE EF EE EE EE EE 00 00 EE EE FF EF FF FF'
 'EE EE EE EE 00 00 EE EE EF EE EF FF EE EE EE EE'
 '00 00 88 88 88 88 88 88 88 88 88 88 00 00 66 66'
 '8A AA A8 77 86 66 66 66 00 00 66 66 8A AA A8 77'
 '86 66 66 66 00 00 66 66 8A AA A8 77 86 66 66 66'
 '00 00 66 66 8A A9 A8 77 86 66 66 66 00 00 66 66'
 '8A A9 A8 77 86 66 66 66 00 00 66 66 8A AA A8 77'
 '86 66 66 66 00 00 66 66 8A AA A8 77 86 66 66 66'
 '00 00 66 66 8A AA A8 77 86 66 66 66 00 00 66 66'
 '8A A8 87 77 86 66 66 66 00 00 66 66 8A 87 77 77'
 '86 66 66 66 00 00 66 66 88 77 77 77 86 66 66 66'
 '00 00 66 66 88 88 88 88 86 66 66 66 00 00 66 66'
 '66 66 66 66 66 66 66 66 00 00 66 66 66 66 66 66'
 '66 66 66 66 00 00'
}


CM_ABOUT BITMAP
{
 '42 4D 66 01 00 00 00 00 00 00 76 00 00 00 28 00'
 '00 00 14 00 00 00 14 00 00 00 01 00 04 00 00 00'
 '00 00 F0 00 00 00 00 00 00 00 00 00 00 00 00 00'
 '00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 80'
 '00 00 00 80 80 00 80 00 00 00 80 00 80 00 80 80'
 '00 00 C0 C0 C0 00 80 80 80 00 00 00 FF 00 00 FF'
 '00 00 00 FF FF 00 FF 00 00 00 FF 00 FF 00 FF FF'
 '00 00 FF FF FF 00 BB BB BB BB BB BB BB BB BB BB'
 '00 00 BB BB BB BB BB BB BB BB BB BB 00 00 BB BB'
 'BB BB BB BB BB BB BB BB 00 00 BB BB BB B8 88 88'
 'BB BB BB BB 00 00 BB BB BB B8 88 88 BB BB BB BB'
 '00 00 BB BB BB B8 00 88 BB BB BB BB 00 00 BB BB'
 'BB B8 00 BB BB BB BB BB 00 00 BB BB BB BB BB BB'
 'BB BB BB BB 00 00 BB BB BB B8 00 88 BB BB BB BB'
 '00 00 BB BB BB BB 80 08 BB BB BB BB 00 00 BB BB'
 'B8 8B B8 00 8B BB BB BB 00 00 BB BB B8 00 BB 00'
 '8B BB BB BB 00 00 BB BB B8 00 BB 00 8B BB BB BB'
 '00 00 BB BB BB 80 00 00 8B BB BB BB 00 00 BB BB'
 'BB B8 00 00 8B BB BB BB 00 00 BB BB BB B8 88 88'
 'BB BB BB BB 00 00 BB BB BB BB BB BB BB BB BB BB'
 '00 00 BB BB BB BB BB BB BB BB BB BB 00 00 BB BB'
 'BB BB BB BB BB BB BB BB 00 00 BB BB BB BB BB BB'
 'BB BB BB BB 00 00'
}

CM_INPUTFILESITEM1 BITMAP "pointfile.bmp"
/*{
 '42 4D 96 01 00 00 00 00 00 00 76 00 00 00 28 00'
 '00 00 18 00 00 00 18 00 00 00 01 00 04 00 00 00'
 '00 00 20 01 00 00 00 00 00 00 00 00 00 00 00 00'
 '00 00 10 00 00 00 00 00 00 00 00 00 80 00 00 80'
 '00 00 00 80 80 00 80 00 00 00 80 00 80 00 80 80'
 '00 00 C0 C0 C0 00 80 80 80 00 00 00 FF 00 00 FF'
 '00 00 00 FF FF 00 FF 00 00 00 FF 00 FF 00 FF FF'
 '00 00 FF FF FF 00 FF FF FF FF FF FF FF FF FF FF'
```

```
 'FF FF FF 00 00 00 00 00 00 00 00 08 8F FF 0A'
 'AA AA AA AA AA AA AA AA 08 8F FF 0A AA AA AA AA'
 'AA AA AA AA 08 8F FF 0A 99 9A AA AA AA AA AA AA'
 '08 8F FF 0A AA AA AA AA AA AA AA AA 08 8F FF 0A'
 'AA AA AA AA AA AA AA AA 08 8F FF 00 00 00 00 00'
 '00 00 00 00 08 8F FF 88 88 88 88 88 88 88 88 88'
 '88 8F FF FF FF FF FF FF FF FF FF FF FF FF FF FF'
 'FF FF FF FF FF FF FF FF FF FF FF F0 00 00 00 00'
 '00 00 0F FF FF FF FF F0 FF FF FF FF FF FF 0F FF'
 'FF FF FF F0 F0 00 FF FF FF FF 0F FF FF FF FF F0'
 'FF FF FF F0 00 0F 0F FF FF FF FF F0 FF FF FF FF'
 'FF FF 0F FF FF FF FF F0 FF FF FF 00 0F FF 0F FF'
 'FF FF FF F0 FF FF FF FF FF F0 8F FF FF FF FF F0'
 'F0 00 FF FF FF 08 8F FF FF FF FF F0 FF FF FF FF'
 '00 88 8F FF FF FF FF F0 FF FF FF F0 88 88 8F FF'
 'FF FF FF F0 00 00 00 08 88 88 8F FF FF FF FF FF'
 'FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF'
 'FF FF FF FF FF FF'
}
*/

CM_POPUPITEM1 BITMAP "tinfile.bmp"
/*{
 '42 4D 96 01 00 00 00 00 00 00 76 00 00 00 28 00'
 '00 00 18 00 00 00 18 00 00 00 01 00 04 00 00 00'
 '00 00 20 01 00 00 00 00 00 00 00 00 00 00 00 00'
 '00 00 10 00 00 00 00 00 00 00 00 00 80 00 00 80'
 '00 00 00 80 80 00 80 00 00 00 80 00 80 00 80 80'
 '00 00 C0 C0 C0 00 80 80 80 00 00 00 FF 00 00 FF'
 '00 00 00 FF FF 00 FF 00 00 00 FF 00 FF 00 FF FF'
 '00 00 FF FF FF 00 FF FF FF FF FF FF FF FF FF FF'
 'FF FF FF 00 00 00 00 00 00 00 00 00 08 8F FF 0A'
 'AA AA AA AA AA AA AA AA 08 8F FF 0A AA AA AA AA'
 'AA AA AA AA 08 8F FF 0A A9 99 9A AA AA AA AA AA'
 '08 8F FF 0A AA AA AA AA AA AA AA AA 08 8F FF 0A'
 'AA AA AA AA AA AA AA AA 08 8F FF 00 00 00 00 00'
 '00 00 00 00 08 8F FF 88 88 88 88 88 88 88 88 88'
 '88 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF'
 'FF FF FF FF FF FF FF FF FF FF FF FF 00 00 00 00'
 '00 00 00 FF FF FF FF FF 0F FF FF FF FF FF F0 FF'
 'FF FF FF FF 0F FF FF FF 00 FF F0 FF FF FF FF FF'
 '0F FF FF FF FF FF F0 FF FF FF FF FF 0F F0 00 0F'
 'FF FF F0 FF FF FF FF FF 0F FF FF FF FF FF F0 FF'
 'FF FF FF FF 0F FF FF F0 0F FF F0 FF FF FF FF FF'
 '0F FF FF FF FF FF 08 FF FF FF FF FF 0F F0 00 FF'
 'FF F0 88 FF FF FF FF FF 0F FF FF FF F0 08 88 FF'
 'FF FF FF FF 0F FF FF FF 08 88 88 FF FF FF FF FF'
 '00 00 00 00 88 88 88 FF FF FF FF FF FF FF FF FF'
 'FF FF FF FF FF FF'
}

*/
```

3D Objects Heights-to-Terrain Interpolation

```
//*********************************************************************//
// Project: Height interpolation from Roof to DTM (TIN)                //
// File: heightinterpol.cpp                                            //
// Input:  .XYZ file, .TIN file, and ROOF file                         //
// Output:  Interpolated Z file                                        //
//                                                                     //
//*********************************************************************//

#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <mem.h>
#include <stdio.h>
#include <string>
#include <conio.h>
#include <math.h>

#define maxpoint 7000
#define maxtriangle 20000
#define accy 1.0e-10

#define max(a, b) ( ((a) > (b)) ? (a) : (b) )
#define min(a, b) ( ((a) < (b)) ? (a) : (b) )

#define max3(a, b, c) ((a) > (b) ? ((a)>(c)?(a):(c)) : ((b)>(c)?(b):(c)))
#define min3(a, b, c) ((a) < (b) ? ((a)<(c)?(a):(c)) : ((b)<(c)?(b):(c)))

class Interpolation
  {
   public:
     struct Point
       {
        float x, y, z;
       };

     struct TIN
       {
        int Node1, Node2, Node3;
       };

     Point* pnt[maxpoint];
     TIN* tin[maxtriangle];


     struct ROOF
       {
        float xr;
        float yr;
        float zr;
       };
     ROOF* roof[maxpoint];
```

```
  typedef struct InterfileStruct
    {
     float xi;
     float yi;
     float zi;
    } Interfile;
  Interfile Element;

  Interpolation() {}
  ~Interpolation() {}

  void ReadXYZ();
  void ReadTIN();
  void ReadROOF();
  void Get3Nodes(int, float&, float&,
                      float&, float&,
                      float&, float&);
  void GetSign(float, float,
               float, float, float, float, float&);
  int FindTri(double, double);
  int PntInTri(double, double,
               double, double,
               double, double,
               double, double);
  void PointInTriangleTest(float, float, bool&);
  void GetPlane(float, float, float, float,
                float, float, float, float, float,
                float&, float&, float&, float&);
  void GetHeightIntersect(float, float, float, float,
                                  float, float, float&);
  float ZInTri(float, float,
               float, float, float,
               float, float, float,
               float, float, float);
  float Rad(float);
  float Deg(float);
  float Max3(float a, float b, float c);
  float Min3(float a, float b, float c);
  int CheckQuadrant(float, float);
  float Azimuth(float, float, float, float);
  void GetAngle(float, float, float, float,
                float, float, float&);
  float Bearing(float, float, float, float);
  void MakeInterpolation();
  void GetIntersectFile();
  void AddZtoFile(Interfile);
  void DeallocateMemory();
  void PressAnyKey();

private:
  int npnt;
  int ntin, t;
  int rpnt, maxroofpnt;
  int Nd1, Nd2, Nd3;
  int Node1, Node2, Node3;
  float xNd1, xNd2, xNd3;
  float yNd1, yNd2, yNd3;
  float zNd1, zNd2, zNd3;
  float zint;
```

```
     bool intri;
     FILE* INTERSECTfile;
     float xroof, yroof, zroof;
     //float xi, yi, zi;
     float a, b, c, d;
  };


void Interpolation :: DeallocateMemory()
  {
   // deallocate memory for XYZ table
   for (int i = 0; i < maxpoint; i ++)
     {
      delete [] pnt[i];
     }

   // deallocate memory for TIN table
   for (int tt = 0; tt < maxtriangle; tt ++)
     {
      delete [] tin[tt];
     }


   // deallocate memory for ROOF table
   for (int r = 0; r < maxpoint; r ++)
     {
      delete [] roof[r];
     }

  }


void Interpolation :: ReadXYZ()
  {
   ifstream XYZfile = "c:\\data\\lochgrd.xyz";

   if (! XYZfile)
      cout << "Unable to open file: XYZ file" << endl;


   // allocate memory for XYZ table
   for (int i =  0; i < maxpoint; i ++)
     {
      pnt[i] = new Point;
     }

   //reads the file header
   string charX, charY, charZ;
   XYZfile >> charX >> charY >> charZ;

   npnt = 0;
   do
     {
      npnt ++;
      XYZfile >> pnt[npnt] -> x
              >> pnt[npnt] -> y
              >> pnt[npnt] -> z;
     } while (! XYZfile.eof());

   XYZfile.close();
  }
```

```
void Interpolation :: ReadTIN()
  {
   ifstream TINfile = "c:\\data\\lochgrd.tin";

   string charNode1, charNode2, charNode3;

   if (! TINfile)
      cout << "Unable to open file: TIN file" << endl;

      // allocate memory for TIN table
      for (int t = 0; t < maxtriangle; t ++)
       {
        tin[t] = new TIN;
       }

      TINfile >> charNode1 >> charNode2 >> charNode3;

      ntin = 0;
      do
        {
         ntin ++;
         TINfile >> tin[ntin] -> Node1
                 >> tin[ntin] -> Node2
                 >> tin[ntin] -> Node3;

        } while (! TINfile.eof());

   TINfile.close();
  }

void Interpolation :: ReadROOF()
  {
   ifstream ROOFfile = "c:\\data\\build3.xyz";

   if (! ROOFfile)
      cout << "Unable to open file: Roof file" << endl;

   // allocate memory for ROOF table
   for (int r =  0; r < maxpoint; r ++)
     {
      roof[r] = new ROOF;
     }

   //reads the file header
   string charX, charY, charZ;
   ROOFfile >> charX >> charY >> charZ;

   rpnt = 0;
   do
     {
      rpnt ++;
      ROOFfile >> roof[rpnt] -> xr
               >> roof[rpnt] -> yr
               >> roof[rpnt] -> zr;
     } while (! ROOFfile.eof());

   maxroofpnt = rpnt;

   ROOFfile.close();
  }
```

```
void Interpolation :: GetIntersectFile()
  {
   char intersectfilename [] = "c:\\data\\lochr2t.xyz";
   INTERSECTfile = fopen(intersectfilename, "w");

   char* charX = "X";
   char* charY = "Y";
   char* charZ = "Z";

   fprintf(INTERSECTfile, "%3s %5s %8s \n", charX, charY, charZ);

   if (INTERSECTfile == NULL)
     {
      cerr << "Could not open INTERSECT file !" << endl;
      exit(1);
     }
   else
     {
      cout << endl;
      cout << "The INTERSECT file is okay for writing." << endl;
     }
  }

void Interpolation :: AddZtoFile(Interfile Int)
  {
   fprintf(INTERSECTfile, "%5.1f %5.1f %5.1f \n", Int.xi, Int.yi, Int.zi);
  }

float Interpolation :: Max3(float a, float b, float c)
  {
   float max;

   max = max(a, b);
   max = max(max, c);
   return max;
  }

float Interpolation :: Min3(float a, float b, float c)
  {
   float min;

   min = min(a, b);
   min = min(min, c);
   return min;
  }

float Interpolation :: Rad(float Deg)
  {
   return M_PI * Deg / 180.0;
  }

float Interpolation :: Deg(float Radian)
  {
   return Radian * 180.0 / M_PI;
  }

float Interpolation :: Bearing(float x1, float y1, float x2, float y2)
  {
   float theta;
```

```
   if (abs(y2 - y1) < accy)
     theta = M_PI / 2;
   else
     theta = atan((x2 - x1) / (y2 - y1));

   return theta;
  }

float Interpolation :: Azimuth(float x1, float y1, float x2, float y2)
  {
   float theta;
   float dx, dy;
   int quadrant;

   dx = x2 - x1;
   dy = y2 - y1;
   if ((dx == 0) && (dy > 0))
     theta = 0;
   else
     if ((dx > 0) && (dy == 0))
       theta = M_PI / 2;
   else
     if ((dx == 0) && (dy < 0))
       theta = M_PI;
   else
     if ((dx < 0) && (dy == 0))
       theta = 3 * M_PI / 2;
   else
     {
      quadrant = CheckQuadrant(dx, dy);

      switch (quadrant)
        {
         case 1:
           theta = Bearing(x1, y1, x2, y2);
           break;

         case 2:
           theta = (2 * M_PI) - abs(Bearing(x1, y1, x2, y2));
           break;

         case 3:
           theta = M_PI + abs(Bearing(x1, y1, x2, y2));
           break;

         case 4:
           theta = M_PI - abs(Bearing(x1, y1, x2, y2));
           break;
        }
     }
   return theta;

  }

void Interpolation :: GetAngle(float xa, float ya, float xb, float yb,
                               float xc, float yc, float& angle)
  {
   bool minus;
   float Az1, Az2;
```

```
   minus = false;
   Az1 = Azimuth(xb, yb, xa, ya);
   Az2 = Azimuth(xb, yb, xc, yc);
   angle = Az2 - Az1;
   if (angle < 0)
     minus = true;
   if (abs(angle) > M_PI)
     {
      angle = (2 * M_PI) - abs(angle);
      if ( ! (minus))
        angle = -angle;  // change direction
     }
  }

void Interpolation :: Get3Nodes(int i, float& xNd1, float& yNd1,
                                       float& xNd2, float& yNd2,
                                       float& xNd3, float& yNd3)
  {
   Nd1 = tin[i]->Node1;
   Nd2 = tin[i]->Node2;
   Nd3 = tin[i]->Node3;

   xNd1 = pnt[Nd1]->x;  // (x, y, z) Node1
   yNd1 = pnt[Nd1]->y;
   zNd1 = pnt[Nd1]->z;

   xNd2 = pnt[Nd2]->x;  // (x, y, z) Node2
   yNd2 = pnt[Nd2]->y;
   zNd2 = pnt[Nd2]->z;

   xNd3 = pnt[Nd3]->x;  // (x, y, z) Node3
   yNd3 = pnt[Nd3]->y;
   zNd3 = pnt[Nd3]->z;
  }

int Interpolation :: CheckQuadrant(float dx, float dy)
  {
   int c;

   if ((dx > 0) && (dy > 0))
     c = 1;
   else
   if ((dx < 0) && (dy > 0))
     c = 2;
   else
   if ((dx < 0) && (dy < 0))
     c = 3;
   if ((dx > 0) && (dy < 0))
     c = 4;

   return c;

  }


void Interpolation :: GetSign(float xr, float yr,
                              float x1, float y1, float x2, float y2, float&
sign)
  {
   float x21, y21;
```

```
   float rsq, rinv;
   float a, b, c;

   x21 = x2 - x1;
   y21 = y2 - y2;
   rsq = (x21 * x21) + (y21 * y21);
   if (rsq < accy)
     cout << "The points coincide" << endl;
   else
     rinv = 1.0 / sqrt(rsq);
     a = -(y21) * rinv;
     b = x21 * rinv;
     c = ((x1*y2) - (x2* y1)) * rinv;
     sign = (a*xr) + (b*yr) + c;
  }


int Interpolation :: PntInTri(double x, double y,
                              double x1, double y1,
                              double x2, double y2,
                              double x3, double y3)
  {
   double o12, o23, o31;
   if (x > max3(x1, x2, x3))
      return 0;
   if (x < min3(x1, x2, x3))
      return 0;
   if (y > max3(y1, y2, y3))
      return 0;
   if (y < min3(y1, y2, y3))
      return 0;

   o12 = x1*y2 - x2*y1 + x2*y - x*y2 + x*y1 - x1*y;
   o23 = x2*y3 - x3*y2 + x3*y - x*y3 + x*y2 - x2*y;
   o31 = x3*y1 - x1*y3 + x1*y - x*y1 + x*y3 - x3*y;

   if      (o12 > 0) return o23 >= 0 && o31 >= 0;
   else if (o12 < 0) return o23 <= 0 && o31 <= 0;
   else
    {
     if      (o23 > 0) return o31 >= 0;
     else if (o23 < 0) return o31 <= 0;
     else              return 1;
    }
  }


int Interpolation :: FindTri(double x, double y)
  {
   int Node1, Node2, Node3, i, found;
   i = 1;
   found = 0;
   while (i <= ntin && !found)
     {
      Node1 = tin[i]->Node1;
      Node2 = tin[i]->Node2;
      Node3 = tin[i]->Node3;
      found = PntInTri(x, y,
                       xNd1, yNd1,
                       xNd2, yNd2,
```

```
                        xNd3, yNd3);
      if (!found)
      i ++;
     }
    return i;
  }

void Interpolation :: PointInTriangleTest(float xr, float yr, bool& intri)
  {
   float angle1, angle2, angle3;
   float SumAng;

   if ((xr == xNd1) && (yr == yNd1))
     {
      zint = zNd1;
      intri = true;
     }
   else
   if ((xr == xNd2) && (yr == yNd2))
     {
      zint = zNd2;
      intri = true;
     }
   else
   if ((xr == xNd3) && (yr == yNd3))
     {
      zint = zNd3;
      intri = true;
     }
   else
     {
      GetAngle(xNd1, yNd1, xr, yr, xNd2, yNd2, angle1);
      GetAngle(xNd2, yNd2, xr, yr, xNd3, yNd3, angle2);
      GetAngle(xNd3, yNd3, xr, yr, xNd1, yNd1, angle3);
      SumAng = angle1 + angle2 + angle3;
      cout << "Sum Angle : " << SumAng << endl;

      if (abs(int(Deg(SumAng))) >= 360)
        intri = true;
      else
      if (int(Deg(SumAng)) == 0)
        intri = false;
      else
        {
         cout << "error ..." << endl;
         cout << "x1: " << xNd1 << " " << "y1: " << yNd1 << " "
              << "Angle1: " << Deg(angle1) << endl;
         cout << "x2: " << xNd2 << " " << "y2: " << yNd2 << " "
              << "Angle2: " << Deg(angle2) << endl;
         cout << "x3: " << xNd3 << " " << "y3: " << yNd3 << " "
              << "Angle3: " << Deg(angle3) << endl;
         cout << "xr: " << xr << " " << "yr: " << yr << " "
              << "Sum Angle: " << Deg(SumAng) << endl;
        }
     }

  }

void Interpolation :: GetPlane(float xj, float yj,
                               float xk, float yk,
```

```
                                float xl, float yl,
                                float zj, float zk, float zl,
                                float& a, float& b, float& c, float& d)
  {
   float xkj, ykj, zkj;
   float xlj, ylj, zlj;

   xkj = xk - xj;
   ykj = yk - yj;
   zkj = zk - zj;
   xlj = xl - xj;
   ylj = yl - yj;
   zlj = zl - zj;

   a = (ykj * zlj) - (zkj * ylj);
   b = (zkj * xlj) - (xkj * zlj);
   c = (xkj * ylj) - (ykj * xlj);
   d = -((xk * a) + (yk * b) + (zk * c));
  }


float Interpolation :: ZInTri(float x, float y,
                              float x1, float y1, float z1,
                              float x2, float y2, float z2,
                              float x3, float y3, float z3)
  {
   float dx, dy, dx2, dy2, dz2, dx3, dy3, dz3, denom;
   dx = x - x1;
   dy = y - y1;
   dx2 = x2 - x1;
   dy2 = y2 - y1;
   dz2 = z2 - z1;
   dx3 = x3 - x1;
   dy3 = y3 - y1;
   dz3 = z3 - z1;
   denom = dx2 * dy3 - dx3 * dy2;
   if (fabs(denom) < 1e-10)
   return z1;

   return z1 - (dx * (dy2*dz3-dy3*dz2) - dy * (dx2*dz3 - dx3*dz2)) / denom;
  }


void Interpolation :: GetHeightIntersect(float xk, float yk, float a, float
b,
                                         float c, float d, float& z)
  {
   float xl, yl, zk, zl, f, g, h, t;
   float denom;
   float x, y;

   xl = xk;
   yl = yk;
   zk = 0;
   zl = 32767;  // false height
   f = xl - xk;
   g = yl - yk;
   h = zl - zk;
   denom = (a*f) + (b*g) + (c*h);
```

```
   if (fabs(denom) < accy)
     cout << "lines and planes are parallel ..." << endl;
   else
     {
      t = -(a*xk + b*yk + c*zk + d) / denom;
      x = xk + (f*t);
      y = yk + (g*t);
      z = zk + (h*t);  // this is the intersected Z and
                       // the triangle plane surface
     }
  }

void Interpolation :: MakeInterpolation()
  {
   //ifstream ROOFfile = "c:\\data\\build2.xyz";

   ReadXYZ();
   ReadTIN();
   ReadROOF();
   GetIntersectFile();

   cout << "Interpolating ....." << endl;
   cout << endl;

   intri = false;
   for (int r = 1; r <= maxroofpnt; r ++)
     {
      xroof = roof[r]->xr;
      yroof = roof[r]->yr;
      //int k = 1;

      for (int k = 1; k < ntin; k ++)
        {
         Get3Nodes(k, xNd1, yNd1, xNd2, yNd2, xNd3, yNd3);

         if ( (xroof >= Min3(xNd1, xNd2, xNd3)) &&
              (xroof <= Max3(xNd1, xNd2, xNd3)) &&
              (yroof >= Min3(yNd1, yNd2, yNd3)) &&
              (yroof <= Max3(yNd1, yNd2, yNd3)) )
            {
             cout << "roof pnt : " << rpnt << endl;
             PointInTriangleTest(xroof, yroof, intri);

             /*t = FindTri(xroof, yroof);
             if (t > ntin)
               zroof = -999;
             else
               {
                Node1 = tin[t]->Node1;
                Node2 = tin[t]->Node2;
                Node3 = tin[t]->Node3;
                zint = ZInTri(xroof, yroof, xNd1, yNd1, zNd1,
                                            xNd2, yNd2, zNd2,
                                            xNd3, yNd3, zNd3);

                Element.xi = xroof;
                Element.yi = yroof;
                Element.zi = zint;
                AddZtoFile(Element);
               }
             */
```

```
             if (intri == true)
               {
                GetPlane(xNd1, yNd1,
                         xNd2, yNd2,
                         xNd3, yNd3,
                         zNd1, zNd2, zNd3,
                         a, b, c, d);
                zint = ZInTri(xroof, yroof, xNd1, yNd1, zNd1,
                                            xNd2, yNd2, zNd2,
                                            xNd3, yNd3, zNd3);
                Element.xi = xroof;
                Element.yi = yroof;
                Element.zi = zint;
                AddZtoFile(Element);
               }
            }
         }
      } // the for loop

   fclose(INTERSECTfile);
  }

void Interpolation :: PressAnyKey()
  {
   cout << endl;
   cout << "Press any key to continue" << endl;
   getch();
   cout << endl;
  }

// main program
void main()
  {
   Interpolation Inter;
   Inter.MakeInterpolation();
   Inter.PressAnyKey();
  }
```

Line Segment-to-Arc/Info Conversion Code

```
//*********************************************************//
//                                                         //
//   A Program to convert SEG file to LIN file (Arc/Info)  //
//      Input: SEG file                                    //
//      Output: LIN file                                   //
//                                                         //
//*********************************************************//

#include <iostream.h>
#include <stdlib.h>
#include <stdio.h>
#include <fstream.h>
#include <conio.h>
#include <math.h>
#include <string>

#define maxpoint 7000

class Seg2Arc
  {
   public:
     float x, y;
     float startx, starty;
     float nextx, nexty;
     int z, nextz, startz;
     int npnt;
     int SegNr, prevSegNr, nextSegNr, startSegNr;

     void MakeArc();

     Seg2Arc(){}
     ~Seg2Arc(){}
  };

// the implementation
void Seg2Arc :: MakeArc()
  {
   string charX, charY, charZ, charSegNr;
   char* charEND = "END";

   ifstream Segmentfile("c:\\data\\cser120.con");
   char Arcfilename [] = "c:\\data\\cser120.lin";

   FILE* Arcfile = fopen(Arcfilename, "w");

   // reads the file header
   Segmentfile >> charX >> charY >> charZ >> charSegNr;
   npnt = 1;
   //prevSegNr = -999;
   Segmentfile >> x >> y >> z >> SegNr;
   startx = x;
```

```
   starty = y;
   startz = z;
   prevSegNr = SegNr;

   while (! Segmentfile.eof())
     {
      printf("\r");
      cout << "Converting to LIN format ...." << npnt;
      Segmentfile >> nextx >> nexty >> nextz >> nextSegNr;

      if (SegNr != prevSegNr)
        {
         fprintf(Arcfile, "%s \n", charEND);
         fprintf(Arcfile, "%5d \n", nextz);
         fprintf(Arcfile, "%3.1f   %3.1f \n", x, y);
         startx = x;
         starty = y;
        }
      else
        {
         if ((x == startx) && (y == starty))
           {
            if (SegNr == prevSegNr)
              {
               fprintf(Arcfile, "%s \n", charEND);
               fprintf(Arcfile, "%5d \n", z);
               fprintf(Arcfile, "%3.1f   %3.1f \n", x, y);
              }
            else
              fprintf(Arcfile, "%3.1f   %3.1f \n", x, y);
           }
         else
         fprintf(Arcfile, "%3.1f   %3.1f \n", x, y);
        }

      prevSegNr = SegNr;
      SegNr = nextSegNr;
      x = nextx;
      y = nexty;
      z = nextz;
      npnt ++;
     }
   fprintf(Arcfile, "%s \n", charEND);
   fprintf(Arcfile, "%s \n", charEND);

   Segmentfile.close();
   fclose(Arcfile);
  }


void main()
  {
   Seg2Arc Arc;
   Arc.MakeArc();
  }
```

3D Viewing Code

```
//*********************************************************//
// A Program to View TIN with arcs and edges on screen      //
// A Program to Display 3D TINs Using OWL                   //
// Input    : (.XYZ), (.TIN), and (.ARC) files              //
// Output   : Windows display                               //
//                                                          //
//*********************************************************//

#include <owl/pch.h>
#include <owl/applicat.h>
#include <owl/framewin.h>
#include <owl/dc.h>
#include <owl/decframe.h>
#include <owl/statusba.h>
#include <owl/controlb.h>
#include <owl/buttonga.h>
#include <owl/gdiobjec.h>
#include <owl/chooseco.h>
#include <owl/inputdia.h>
#include <owl/opensave.h>

#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <stdio.h>
#include <stddef.h>
#include <math.h>
#include <mem.h>
#include <string.h>

#include "view3d.rh"


// constants for maxpoints, tins, and arcs
const int maxtin = 2000;
const int maxpoint = 700;
const int maxarc = 200;
const int maxsegment = 1000;

// view parameters constants

const double theta = -60.0;  // was -60.0
const double phi = 60.0;
const double ViewPlaneDist = 2000.0; // 500 for the trylake
                                     // the bigger the ViewPlaneDist
                                     // -> larger view
const double rho = 2500.0;           // 1500 for the trylake

class TView3DWindowApp : public TApplication
  {
   public:
```

```
      TView3DWindowApp() : TApplication() {}  // call base class constructor
      void InitMainWindow();
  };


class TView3D : public TWindow
  {
   public:
      TView3D(TWindow* parent = 0)
              : TWindow(parent) {}

      typedef double Matrix[4][4];
      struct Vector
        {
         double  x, y, z;
        };

      struct Point
        {
         double x, y, z;
        };

      struct Pointt
        {
         float x, y, z;
        };

      struct TIN
        {
         int Node1, Node2, Node3;
        };

      struct TIN2
        {
         int Node1, Node2, Node3;
        };

      Point* pnt[maxpoint];
      Pointt* pntt[maxpoint];
      TIN* tin[maxtin];
      TIN2* tinn[maxtin];

      struct Polygon
        {
         int Snode;
         int Enode;
        };
      Polygon* poly_ptr[maxarc];

      struct RoofArc
        {
         int Snode;
         int Enode;
        };
      RoofArc* roofarc[maxarc];

      FILE* ARCfile;
      FILE* ARCRooffile;

      // struct for trees
      struct TreePoints
```

```
  {
   double x;
   double y;
   double z;
  };
TreePoints* treepnts[maxpoint];

struct TreeArcs
  {
   int Snode;
   int Enode;
  };
TreeArcs* treearc[maxarc];

// file for the tree points and arcs
FILE* Treepntfile;
FILE* Treearcfile;

struct Segment
  {
   double x;
   double y;
   int Hreq;
   int SegNr;
  };
Segment* Seg[maxsegment];

int i, t, tt, k;
int npnt, ntin, narc, n;
int npntt, ntinn, narcr;
int ntree, narct;
string charNode1, charNode2, charNode3;
double xmin, xmax, ymin, ymax;
double zmin, zmax;
double xlength, ylength, zlength;

double startx, starty;
double nextx, nexty;
int SegNr, nextSegNr, prevSegNr;
double x, y;
int h, nexth;

double MaxX, MaxY;
double xmid, ymid;
double screen_x, screen_y;
double scaleX, scaleY, scaleZ;
double sin_theta;
double cos_theta;
double sin_phi;
double cos_phi;
double x_view, y_view, z_view;
Matrix viewT;
Vector VecWorld, VecView;
double DotProduct(Vector, Vector);
double Magnitude(Vector);
void Transform(int, float&, float&);
void Normalize(Vector, Vector&);
void SetZeroMatrix(Matrix&);
void SetIdentityMatrix(Matrix&);
void VectorMatrix(Vector, Matrix, Vector&);
```

```
      void SetViewTransformationMatrix(Matrix);
      void SetViewVariables(double&, double&, double&);
      void ReadXYZ();
      void ReadXYZ2();
      void ReadTIN();
      void ReadTIN2();
      void ReadARC();
      void ReadRoofArc();
      void ReadTreeXYZ();
      void ReadTreeArc();
      void ReadSegment();
      void DeallocateMemory();
      void GetMinMax(double&, double&, double&, double&, double&, double&,
                     double&, double&, double&);
      void Perspective(double, double, double, double&, double&);
      double InRadians(double);
      void Paint(TDC& dc, bool, TRect&);
      void EvSize(uint, TSize&);
      void CmInputFilesItem1();
      void CmPopupItem1();
      void CmFileExit()
           {PostQuitMessage(0);}
      void CmClear();
      void CmAbout();

      ~TView3D() {}

      DECLARE_RESPONSE_TABLE(TView3D);
   };

DEFINE_RESPONSE_TABLE1(TView3D, TWindow)
   EV_WM_SIZE,
   EV_COMMAND(CM_INPUTFILESITEM1, CmInputFilesItem1),
   EV_COMMAND(CM_POPUPITEM1, CmPopupItem1),
   EV_COMMAND(CM_ABOUT, CmAbout),
   EV_COMMAND(CM_FILEEXIT, CmFileExit),
   EV_COMMAND(CM_CLEAR, CmClear),
END_RESPONSE_TABLE;


// the definitions section
void TView3D :: DeallocateMemory()
  {
   // for XYZ table
   for (i = 0; i < maxpoint; i ++)
     {
      delete [] pnt[i];
     }

   // for XYZ2 table
   for (int ii = 0; ii < maxpoint; ii ++)
     {
      delete [] pntt[ii];
     }

   // for TIN table
   for (t = 0; t < maxtin; t ++)
     {
      delete [] tin[t];
     }
```

```
   // for TIN2 table
   for (int tt = 0; tt < maxtin; tt ++)
     {
      delete [] tinn[tt];
     }

   for (int ii = 0; ii < maxarc; ii ++)
     {
      delete [] poly_ptr[ii];
     }

   for (int ir = 0; ir < maxarc; ir ++)
     {
      delete [] roofarc[ir];
     }

   // for the tree points
   for (int tp = 0; tp < maxpoint; tp ++)
     {
      delete [] treepnts[tp];
     }

   // for the tree arcs
   for (int ta = 0; ta < maxarc; ta ++)
     {
      delete [] treearc[ta];
     }

   // for the segment contours
   for (int sg = 0; sg < maxsegment; sg ++)
     {
      delete [] Seg[sg];
     }

  }

void TView3D :: ReadXYZ()
  {
   //ifstream XYZfile = "c:\\data\\lakebore.xyz";
   //ifstream XYZfile = "c:\\data\\test.xyz";
   ifstream XYZfile = "c:\\data\\lochass.xyz";

   if (! XYZfile)
      MessageBox ("Unable to open file: XYZ file", "File Error",
                     MB_OK | MB_ICONEXCLAMATION);

   // for XYZ table
   for (i =  0; i < maxpoint; i ++)
     {
      pnt[i] = new Point;
     }

   //reads the file header
   string charX, charY, charZ;
   XYZfile >> charX >> charY >> charZ;

   npnt = 0;
   do
     {
      npnt ++;
      XYZfile >> pnt[npnt] -> x
```

```
              >> pnt[npnt] -> y
              >> pnt[npnt] -> z;
     } while (! XYZfile.eof());

   XYZfile.close();
  }

void TView3D :: ReadXYZ2()
  {
   //ifstream XYZ2file = "c:\\data\\test400.xyz";
   //ifstream XYZ2file = "c:\\data\\lake200.xyz";
   //ifstream XYZ2file = "c:\\data\\testrnf.xyz";
   ifstream XYZ2file = "c:\\data\\lochfpr.xyz";

   if (! XYZ2file)
      MessageBox ("Unable to open file: XYZ file", "File Error",
                    MB_OK | MB_ICONEXCLAMATION);

   // for XYZ table
   for (i =  0; i < maxpoint; i ++)
     {
      pntt[i] = new Pointt;
     }

   //reads the file header
   string charX, charY, charZ;
   XYZ2file >> charX >> charY >> charZ;

   npntt = 0;
   do
     {
      npntt ++;
      XYZ2file >> pntt[npntt] -> x
               >> pntt[npntt] -> y
               >> pntt[npntt] -> z;
     } while (! XYZ2file.eof());

   XYZ2file.close();
  }



void TView3D :: ReadTIN()
  {
   string charNode1, charNode2, charNode3;

   //ifstream TINfile = "c:\\data\\lakebo.tin";
   //ifstream TINfile = "c:\\data\\testr.tin";
   ifstream TINfile = "c:\\data\\lochas.tin";

   if (! TINfile)
      MessageBox("Unable to open file: TIN file",
                 "File Error", MB_OK | MB_ICONEXCLAMATION);

      // for TIN table
      for (t = 0; t < maxtin; t ++)
       {
        tin[t] = new TIN;
       }

      TINfile >> charNode1 >> charNode2 >> charNode3;
```

```
      ntin = 0;
      do
        {
         ntin ++;
         TINfile >> tin[ntin] -> Node1
                 >> tin[ntin] -> Node2
                 >> tin[ntin] -> Node3;

        } while (! TINfile.eof());

   TINfile.close();
  }

void TView3D :: ReadTIN2()
  {
   //ifstream TIN2file = "c:\\data\\linetin.tin";
   ifstream TIN2file = "c:\\data\\lake2ft.tin";

   if (! TIN2file)
      MessageBox("Unable to open file: TIN file",
                 "File Error", MB_OK | MB_ICONEXCLAMATION);

      // for TIN2 table
      for (tt = 0; tt < maxtin; tt ++)
       {
        tinn[tt] = new TIN2;
       }

      TIN2file >> charNode1 >> charNode2 >> charNode3;
      ntinn = 0;
      do
        {
         ntinn ++;
         TIN2file >> tinn[ntinn] -> Node1
                  >> tinn[ntinn] -> Node2
                  >> tinn[ntinn] -> Node3;

        } while (! TIN2file.eof());

   TIN2file.close();
  }



void TView3D :: ReadARC()
  {
   string charSnode, charEnode;

   // Reads again the the XYZ file
   //ifstream ARCfile = "c:\\data\\lake.arc";
   //ifstream ARCfile = "c:\\data\\test.arc";
   ifstream ARCfile = "c:\\data\\lake.arc";

   // Allocate memory for Arcs
   for (int ii = 0; ii < maxarc; ii ++)
     {
      poly_ptr[ii] = new Polygon;
     }

   if (! ARCfile)
     {
```

```
      MessageBox ("Unable to open file: ARC file", "FileError",
                  MB_OK | MB_ICONEXCLAMATION);
    }
   else
    {
     // reads the file header
     ARCfile >> charSnode >> charEnode;
     narc = 0;
       do
        {
         narc ++;
         ARCfile >> poly_ptr[narc] -> Snode
                 >> poly_ptr[narc] -> Enode;

        } while (! ARCfile.eof());

    //if (ARCfile.eof() == true)
       //MessageBox ("Finished read the ARC points", "ARC file complete",
                   //MB_OK | MB_ICONINFORMATION);
    }
   ARCfile.close();
  }

void TView3D :: ReadRoofArc()
  {
   string charSnode, charEnode;

   //ifstream ARCRooffile = "c:\\data\\roofarc.arc";
   ifstream ARCRooffile = "c:\\data\\lochbld.arc";

   // Allocate memory for Arcs
   for (int ii = 0; ii < maxarc; ii ++)
    {
     roofarc[ii] = new RoofArc;
    }

   if (! ARCRooffile)
    {
     MessageBox ("Unable to open file: ARC roof file", "FileError",
                  MB_OK | MB_ICONEXCLAMATION);
    }
   else
    {
     // reads the file header
     ARCRooffile >> charSnode >> charEnode;
     narcr = 0;
       do
        {
         narcr ++;
         ARCRooffile >> roofarc[narcr] -> Snode
                     >> roofarc[narcr] -> Enode;

        } while (! ARCRooffile.eof());

    //if (ARCfile.eof() == true)
       //MessageBox ("Finished read the ARC points", "ARC file complete",
                   //MB_OK | MB_ICONINFORMATION);
    }
   ARCRooffile.close();
  }
```

```
// for the tree points
void TView3D :: ReadTreeXYZ()
  {
   ifstream Treepntfile = "c:\\data\\tree1.xyz";

   if (! Treepntfile)
      MessageBox ("Unable to open file: XYZ file", "File Error",
                    MB_OK | MB_ICONEXCLAMATION);

   // for tree XYZ table
   for (i =  0; i < maxpoint; i ++)
     {
      treepnts[i] = new TreePoints;
     }

   //reads the file header
   string charX, charY, charZ;
   Treepntfile >> charX >> charY >> charZ;

   ntree = 0;
   do
     {
      ntree ++;
      Treepntfile >> treepnts[ntree] -> x
                  >> treepnts[ntree] -> y
                  >> treepnts[ntree] -> z;
     } while (! Treepntfile.eof());

   Treepntfile.close();
  }

void TView3D :: ReadTreeArc()
  {
   string charSnode, charEnode;

   ifstream Treearcfile = "c:\\data\\tree1.arc";

   // Allocate memory for tree arcs
   for (int ii = 0; ii < maxarc; ii ++)
     {
      treearc[ii] = new TreeArcs;
     }

   if (! Treearcfile)
     {
      MessageBox ("Unable to open file: Tree arc file", "FileError",
                   MB_OK | MB_ICONEXCLAMATION);
     }
   else
     {
      // reads the file header
      Treearcfile >> charSnode >> charEnode;
      narct = 0;
        do
         {
          narct ++;
          Treearcfile >> treearc[narct] -> Snode
                      >> treearc[narct] -> Enode;

         } while (! Treearcfile.eof());
```

```
     //if (ARCfile.eof() == true)
        //MessageBox ("Finished read the ARC points", "ARC file complete",
                    //MB_OK | MB_ICONINFORMATION);
     }
    Treearcfile.close();
   }

/*void TView3D :: ReadSegment()
   {
    string charX, charY, charHreq, charSegNr;

    // to handle the contour segments


    //ifstream Segmentfile("c:\\data\\pent.con");
    //ifstream Segmentfile("c:\\data\\lochgrd4.con");
    //ifstream Segmentfile("c:\\data\\sercom30.con");
    //ifstream Segmentfile("c:\\data\\sercom50.con");
    //ifstream Segmentfile("c:\\data\\sercom10.con");
    ifstream SegmentFile = "c:\\data\\lochgrd.con";

    // Allocate memory for tree arcs
    for (int sg = 0; sg < maxsegment; sg ++)
      {
       Seg[sg] = new Segment;
      }

    if (! SegmentFile)
      {
       MessageBox ("Unable to open file: Segment contours file", "FileError",
                    MB_OK | MB_ICONEXCLAMATION);
      }
    else
      {
       // reads the header for the segment file
       SegmentFile >> charX >> charY >> charHreq >> charSegNr;
       prevSegNr = -999;
       SegmentFile >> x >> y >> h >> SegNr;
       startx = x;
       starty = y;

       int nseg = 0;
         do
          {
           nseg ++;
           SegmentFile >> Seg[nseg]->x
                       >> Seg[nseg]->y
                       >> Seg[nseg]->Hreq
                       >> Seg[nseg]->SegNr;

          } while (! SegmentFile.eof());

     //if (ARCfile.eof() == true)
        //MessageBox ("Finished read the ARC points", "ARC file complete",
                    //MB_OK | MB_ICONINFORMATION);
     }
    SegmentFile.close();
   }
*/

void TView3D :: Transform(int MaxY, float& x, float& y)
```

```
  {
   x = (x - xmin) * scaleX;
   y = MaxY - ((y - ymin) * scaleY);
  }

double TView3D :: DotProduct(Vector v1, Vector v2)
  {
   double dp;
   dp = (v1.x * v2.x) + (v1.y * v2.y) + (v1.z * v2.z);
   return dp;
  }

double TView3D :: Magnitude(Vector v)
  {
   double Mag;
   Mag = sqrt((v.x * v.x) + (v.y * v.y) + (v.z * v.z));
   return Mag;
  }

void TView3D :: Normalize(Vector v1, Vector& v2)
  {
   double denominator;

   denominator = Magnitude(v1);
   if (denominator == 0)
      v2 = v1;
   else
     {
      v2.x = v1.x / denominator;
      v2.y = v1.y / denominator;
      v2.z = v1.z / denominator;
     }
  }

void TView3D :: VectorMatrix(Vector v1, Matrix T, Vector& v2)
  {
   T[3][2] = rho;
   T[3][3] = 1;
   v2.x = v1.x * T[0][0] + v1.y * T[1][0] + v1.z * T[2][0] + T[3][0];
   v2.y = v1.x * T[0][1] + v1.y * T[1][1] + v1.z * T[2][1] + T[3][1];
   v2.z = v1.x * T[0][2] + v1.y * T[1][2] + v1.z * T[2][2] + T[3][2];
  }

void TView3D :: SetZeroMatrix(Matrix& m)
  {
   for (int i = 0; i < 3; i ++)
     for (int j = 0; j < 3; j ++)
       {
        m[i][j] = 0;
       }
  }

void TView3D :: SetIdentityMatrix(Matrix& m)
  {
   SetZeroMatrix(m);
   for (int diag = 0; diag < 3; diag ++)
     for (diag = 0; diag < 3; diag ++)
       {
        m[diag][diag] = 1;
       }
  }
```

```
double TView3D :: InRadians(double degrees)
  {
   return degrees * M_PI / 180;
  }

void TView3D :: SetViewVariables(double& x_view, double& y_view, double&
z_view)
  {
   sin_theta = sin((InRadians(theta)));
   cos_theta = cos((InRadians(theta)));
   sin_phi   = sin((InRadians(phi)));
   cos_phi   = cos((InRadians(phi)));
   x_view = rho * cos_theta * sin_phi;
   y_view = rho * sin_theta * sin_phi;
   z_view = rho * cos_phi;
  }

void TView3D :: SetViewTransformationMatrix(Matrix viewT)
  {
   viewT[0][0] = - sin_theta;
   viewT[0][1] = - cos_theta * cos_phi;
   viewT[0][2] = - cos_theta * sin_phi;
   viewT[1][0] =   cos_theta;
   viewT[1][1] = - sin_theta * cos_phi;
   viewT[1][2] = - sin_theta * sin_phi;
   viewT[2][1] =   sin_phi;
   viewT[2][2] = - cos_phi;
   viewT[3][2] =   rho;
   viewT[3][3] = 1;
  }

void TView3D :: Perspective(double xx, double yy, double zz,
                            double& screen_x, double& screen_y)
  {
   double temp_x, temp_y;

   xmid = (MaxX + 1) / 2;
   ymid = (MaxY + 1) / 2;
   temp_x = ViewPlaneDist * (xx / zz);
   temp_y = ViewPlaneDist * (yy / zz);
   screen_x = xmid + temp_x;
   screen_y = ymid + temp_y;
  }

void TView3D :: GetMinMax(double& xmax, double& ymax, double& zmax,
                          double& xmin, double& ymin, double& zmin,
                          double& xlength, double& ylength, double& zlength)
  {
   //ifstream XYZfile = "c:\\data\\test.xyz";
   ifstream XYZfile = "c:\\data\\lochass.xyz";

   if (! XYZfile)
      MessageBox ("Unable to open file: XYZ file", "FileError",
                     MB_OK | MB_ICONEXCLAMATION);

   string charX, charY, charZ;
   double xa, ya, za;
   // reads the file header
   XYZfile >> charX >> charY >> charZ;

   XYZfile >> xa >> ya >> za;
```

```
      xmin = xa;
      xmax = xa;

      ymin = ya;
      ymax = ya;

      zmin = za;
      zmax = za;

      do
        {
         XYZfile >> xa >> ya >> za;

         if (xmin > xa)
             xmin = xa;

         if (ymin > ya)
             ymin = ya;

         if (zmin > za)
             zmin = za;

         if (xmax < xa)
             xmax = xa;

         if (ymax < ya)
             ymax = ya;

         if (zmax < za)
             zmax = za;
         } while (! XYZfile.eof());

      xlength = xmax - xmin;
      ylength = ymax - ymin;
      zlength = zmax - zmin;
  }

void TView3D :: Paint(TDC& dc, bool, TRect&)
  {
   GetMinMax(xmax, ymax, zmax, xmin, ymin, zmin,
             xlength, ylength, zlength);

   ReadXYZ();
   ReadXYZ2();
   ReadTIN();
   //ReadTIN2();
   ReadARC();      // reads the ARC
   ReadRoofArc();  // the roof file
   ReadTreeXYZ();  // the tree points
   ReadTreeArc();  // the tree arcs
   //ReadSegment();  // the contour segments

   // to set the backgriund color
   static const TColor color (RGB(0, 0, 0));  // RGB - black
   TWindow :: SetBkgndColor(color);

   TRect rect;
   :: GetClientRect(HWindow, &rect);

   // set the origin in the middle of the window client area
   dc.SetMapMode(MM_ANISOTROPIC);
```

```
/*dc.SetWindowExt(TSize((xmax-xmin), (ymax-ymin)));
dc.SetWindowOrg(TPoint(xmin, ymin));
dc.SetViewportExt(GetClientRect().Size());
dc.SetViewportOrg(TPoint(0, 0));
*/

SetViewportOrgEx(dc, rect.left-180, rect.top-300, &POINT());
// the above parameter is for the test data

// SetViewportOrgEx(dc, rect.left, rect.top, &POINT());
// the above parameter for the trylake.xyz data

//SetViewportOrgEx(dc, rect.right/2, rect.top+20, &POINT());

scaleX = (rect.right - rect.left) / xlength;
scaleY = (rect.bottom - rect.top) / ylength;

SetViewVariables(x_view, y_view, z_view);
SetViewTransformationMatrix(viewT);

// Select pen style for TENs points
TPen pen3 = TPen(RGB(250, 0, 0), 3, PS_SOLID);
SelectObject(dc, pen3);

for (k = 1; k < npnt; k ++)
  {
   VecWorld.x = pnt[k]->x;
   VecWorld.y = pnt[k]->y;
   VecWorld.z = pnt[k]->z;

   VectorMatrix(VecWorld, viewT, VecView);

   pnt[k]->x = VecView.x;
   pnt[k]->y = VecView.y;
   pnt[k]->z = VecView.z;

   pnt[k]->x = (pnt[k]->x - xmin)* scaleX;
   pnt[k]->y = (rect.bottom - rect.top) - ((pnt[k]->y - ymin) * scaleY);

   Perspective(pnt[k]->x, pnt[k]->y, pnt[k]->z, screen_x, screen_y);

   pnt[k]->x = screen_x;
   pnt[k]->y = screen_y;

   // to draw TENs points
   dc.MoveTo(pnt[k]->x, pnt[k]->y);
   dc.LineTo(pnt[k]->x, pnt[k]->y);

  }
SelectObject(dc, pen3);
DeleteObject(pen3);


// to handle points for TINS
// Select pen style for points
/*TPen pentinpnt = TPen(RGB(10, 10, 10), 3, PS_SOLID);
SelectObject(dc, pentinpnt);
for (int kk = 1; kk < npntt; kk ++)
  {
   VecWorld.x = pntt[kk]->x;
   VecWorld.y = pntt[kk]->y;
```

```
      VecWorld.z = pntt[kk]->z;

      VectorMatrix(VecWorld, viewT, VecView);

      pntt[kk]->x = VecView.x;
      pntt[kk]->y = VecView.y;
      pntt[kk]->z = VecView.z;

      pntt[kk]->x = (pntt[kk]->x - xmin)* scaleX;
        pntt[kk]->y = (rect.bottom - rect.top) - ((pntt[kk]->y - ymin) *
scaleY);

     Perspective(pntt[kk]->x, pntt[kk]->y, pntt[kk]->z, screen_x, screen_y);

      pntt[kk]->x = screen_x;
      pntt[kk]->y = screen_y;

      // to draw points
      dc.MoveTo(pntt[kk]->x, pntt[kk]->y);
      dc.LineTo(pntt[kk]->x, pntt[kk]->y);

     }
   SelectObject(dc, pentinpnt);
   DeleteObject(pentinpnt);
   */

   // set pen style for polygon (i.e. constrained edges) display
   TPen penpoly = TPen(RGB(255, 0, 0), 2, PS_SOLID);
   SelectObject(dc, penpoly);

   /*for (int r = 1; r < narc; r ++)
     {
      dc.MoveTo(pntt[poly_ptr[r]->Snode]->x,
                pntt[poly_ptr[r]->Snode]->y);
      dc.LineTo(pntt[poly_ptr[r]->Enode]->x,
                pntt[poly_ptr[r]->Enode]->y);
      dc.LineTo(pntt[poly_ptr[r]->Snode]->x,
                pntt[poly_ptr[r]->Snode]->y);
     }
    */
   for (int r = 1; r < narc; r ++)
     {
      dc.MoveTo(pnt[poly_ptr[r]->Snode]->x,
                pnt[poly_ptr[r]->Snode]->y);
      dc.LineTo(pnt[poly_ptr[r]->Enode]->x,
                pnt[poly_ptr[r]->Enode]->y);
      dc.LineTo(pnt[poly_ptr[r]->Snode]->x,
                pnt[poly_ptr[r]->Snode]->y);
     }

   SelectObject(dc, penpoly);
   DeleteObject(penpoly);


   // Select pen style (different colour)
   TPen pen1 = TPen(RGB(255, 255, 255), 1, PS_SOLID);
   SelectObject(dc, pen1);

   /*
   // Display the 8 corners of the 3D space
   dc.MoveTo(pnt[1]->x, pnt[1]->y);
```

```
dc.LineTo(pnt[2]->x, pnt[2]->y);
dc.LineTo(pnt[3]->x, pnt[3]->y);
dc.LineTo(pnt[4]->x, pnt[4]->y);
dc.LineTo(pnt[1]->x, pnt[1]->y);
dc.LineTo(pnt[5]->x, pnt[5]->y);
dc.LineTo(pnt[6]->x, pnt[6]->y);
dc.LineTo(pnt[7]->x, pnt[7]->y);
dc.LineTo(pnt[8]->x, pnt[8]->y);
dc.LineTo(pnt[5]->x, pnt[5]->y);
dc.MoveTo(pnt[3]->x, pnt[3]->y);
dc.LineTo(pnt[7]->x, pnt[7]->y);
dc.MoveTo(pnt[2]->x, pnt[2]->y);
dc.LineTo(pnt[6]->x, pnt[6]->y);
dc.MoveTo(pnt[4]->x, pnt[4]->y);
dc.LineTo(pnt[8]->x, pnt[8]->y);

SelectObject(dc, pen1);
DeleteObject(pen1);
*/

// to handle TEN
// Another pen style (different colour)
TPen penten = TPen(RGB(0, 255, 0), 1, PS_SOLID);
SelectObject(dc, penten);

for (k = 1; k < ntin; k ++)
  {
   dc.MoveTo(pnt[tin[k]->Node1]->x,
             pnt[tin[k]->Node1]->y);
   dc.LineTo(pnt[tin[k]->Node2]->x,
             pnt[tin[k]->Node2]->y);
   dc.LineTo(pnt[tin[k]->Node3]->x,
             pnt[tin[k]->Node3]->y);
   dc.LineTo(pnt[tin[k]->Node1]->x,
             pnt[tin[k]->Node1]->y);
  }

SelectObject(dc, penten);
DeleteObject(penten);

/*
// to handle TIN
TPen pentintop = TPen(RGB(255, 0, 0), 1, PS_SOLID);
SelectObject(dc, pentintop);

for (k = 1; k < ntinn; k ++)
  {
   dc.MoveTo(pntt[tinn[k]->Node1]->x,
             pntt[tinn[k]->Node1]->y);
   dc.LineTo(pntt[tinn[k]->Node2]->x,
             pntt[tinn[k]->Node2]->y);
   dc.LineTo(pntt[tinn[k]->Node3]->x,
             pntt[tinn[k]->Node3]->y);
   dc.LineTo(pntt[tinn[k]->Node1]->x,
             pntt[tinn[k]->Node1]->y);
  }

SelectObject(dc, pentintop);
DeleteObject(pentintop);
*/
```

```
/*
// to handle boreholes
TPen penbore = TPen(RGB(255, 255, 10), 1, PS_SOLID);
SelectObject(dc, penbore);

// this is for lakebore data
dc.MoveTo(pnt[9]->x, pnt[9]->y);
dc.LineTo(pnt[10]->x, pnt[10]->y);
dc.LineTo(pnt[11]->x, pnt[11]->y);
dc.LineTo(pnt[12]->x, pnt[12]->y);
dc.LineTo(pnt[13]->x, pnt[13]->y);
dc.LineTo(pnt[14]->x, pnt[14]->y);

dc.MoveTo(pnt[15]->x, pnt[15]->y);
dc.LineTo(pnt[16]->x, pnt[16]->y);
dc.LineTo(pnt[17]->x, pnt[17]->y);
dc.LineTo(pnt[18]->x, pnt[18]->y);
dc.LineTo(pnt[19]->x, pnt[19]->y);
dc.LineTo(pnt[20]->x, pnt[20]->y);

dc.MoveTo(pnt[21]->x, pnt[21]->y);
dc.LineTo(pnt[22]->x, pnt[22]->y);
dc.LineTo(pnt[23]->x, pnt[23]->y);
dc.LineTo(pnt[24]->x, pnt[24]->y);
dc.LineTo(pnt[25]->x, pnt[25]->y);
dc.LineTo(pnt[26]->x, pnt[26]->y);

dc.MoveTo(pnt[27]->x, pnt[27]->y);
dc.LineTo(pnt[28]->x, pnt[28]->y);
dc.LineTo(pnt[29]->x, pnt[29]->y);
dc.LineTo(pnt[30]->x, pnt[30]->y);
dc.LineTo(pnt[31]->x, pnt[31]->y);
dc.LineTo(pnt[32]->x, pnt[32]->y);

dc.MoveTo(pnt[33]->x, pnt[33]->y);
dc.LineTo(pnt[34]->x, pnt[34]->y);
dc.LineTo(pnt[35]->x, pnt[35]->y);
dc.LineTo(pnt[36]->x, pnt[36]->y);
dc.LineTo(pnt[37]->x, pnt[37]->y);
dc.LineTo(pnt[38]->x, pnt[38]->y);

dc.MoveTo(pnt[39]->x, pnt[39]->y);
dc.LineTo(pnt[40]->x, pnt[40]->y);
dc.LineTo(pnt[41]->x, pnt[41]->y);
dc.LineTo(pnt[42]->x, pnt[42]->y);
dc.LineTo(pnt[43]->x, pnt[43]->y);
dc.LineTo(pnt[44]->x, pnt[44]->y);

SelectObject(dc, penbore);
DeleteObject(penbore);
*/

// to handle buildings
// Select pen style for building roof points
TPen penbuildpnt = TPen(RGB(250, 0, 0), 3, PS_SOLID);
SelectObject(dc, penbuildpnt);

for (int k = 1; k < npntt; k ++)
  {
   VecWorld.x = pntt[k]->x;
   VecWorld.y = pntt[k]->y;
```

```
    VecWorld.z = pntt[k]->z;

    VectorMatrix(VecWorld, viewT, VecView);

    pntt[k]->x = VecView.x;
    pntt[k]->y = VecView.y;
    pntt[k]->z = VecView.z;

    pntt[k]->x = (pntt[k]->x - xmin)* scaleX;
   pntt[k]->y = (rect.bottom - rect.top) - ((pntt[k]->y - ymin) * scaleY);

    Perspective(pntt[k]->x, pntt[k]->y, pntt[k]->z, screen_x, screen_y);

    pntt[k]->x = screen_x;
    pntt[k]->y = screen_y;

    // to draw building roof points
    dc.MoveTo(pntt[k]->x, pntt[k]->y);
    dc.LineTo(pntt[k]->x, pntt[k]->y);
   }
SelectObject(dc, penbuildpnt);
DeleteObject(penbuildpnt);

// set pen style for the building roof lines
TPen penbuildline = TPen(RGB(0, 255, 255), 2, PS_SOLID);
SelectObject(dc, penbuildline);

for (int f = 1; f < narcr; f ++)
  {
   dc.MoveTo(pntt[roofarc[f]->Snode]->x,
             pntt[roofarc[f]->Snode]->y);
   dc.LineTo(pntt[roofarc[f]->Enode]->x,
             pntt[roofarc[f]->Enode]->y);
   dc.LineTo(pntt[roofarc[f]->Snode]->x,
             pntt[roofarc[f]->Snode]->y);
  }

SelectObject(dc, penbuildline);
DeleteObject(penbuildline);


// to handle trees
// Select pen style for trees points
TPen pentreepnt = TPen(RGB(250, 0, 0), 2, PS_SOLID);
SelectObject(dc, pentreepnt);

for (int kt = 1; kt < ntree; kt ++)
  {
   VecWorld.x = treepnts[kt]->x;
   VecWorld.y = treepnts[kt]->y;
   VecWorld.z = treepnts[kt]->z;

   VectorMatrix(VecWorld, viewT, VecView);

   treepnts[kt]->x = VecView.x;
   treepnts[kt]->y = VecView.y;
   treepnts[kt]->z = VecView.z;

   treepnts[kt]->x = (treepnts[kt]->x - xmin)* scaleX;
   treepnts[kt]->y = (rect.bottom - rect.top) - ((treepnts[kt]->y - ymin)
                                                 * scaleY);
```

```
    Perspective(treepnts[kt]->x, treepnts[kt]->y, treepnts[kt]->z, screen_x,
screen_y);

      treepnts[kt]->x = screen_x;
      treepnts[kt]->y = screen_y;

      // to draw building roof points
      dc.MoveTo(treepnts[kt]->x, treepnts[kt]->y);
      dc.LineTo(treepnts[kt]->x, treepnts[kt]->y);
     }
   SelectObject(dc, pentreepnt);
   DeleteObject(pentreepnt);

   // set pen style for the tree arcs
   TPen pentreearcs = TPen(RGB(255, 0, 0), 2, PS_SOLID);
   SelectObject(dc, pentreearcs);

   for (int ft = 1; ft < narct; ft ++)
     {
      dc.MoveTo(treepnts[treearc[ft]->Snode]->x,
                treepnts[treearc[ft]->Snode]->y);
      dc.LineTo(treepnts[treearc[ft]->Enode]->x,
                treepnts[treearc[ft]->Enode]->y);
      dc.LineTo(treepnts[treearc[ft]->Snode]->x,
                treepnts[treearc[ft]->Snode]->y);
     }

   SelectObject(dc, pentreearcs);
   DeleteObject(pentreearcs);

   // to handle the contour segments
   // set pen style for the segments

   // colour guides:
   // Light Magenta: (255 0 255)
   // Light Yellow : (255, 255, 0)
   // Light Grey   : (192, 192, 192)
   // Grey         : (128, 128, 128)
   // Light Cyan   : (0, 255, 255)

   TPen pensegment = TPen(RGB(255, 255, 0), 2, PS_SOLID);
   SelectObject(dc, pensegment);

   // to handle the contour segments

   //ifstream SegmentFile("c:\\data\\pent.con");
   ifstream SegmentFile("c:\\data\\lochgrd4.con");
   //ifstream SegmentFile("c:\\data\\sercom30.con");
   //ifstream SegmentFile("c:\\data\\sercom50.con");
   //ifstream SegmentFile("c:\\data\\sercom10.con");
   //ifstream SegmentFile = "c:\\data\\lochgrd.con";

   string charX, charY, charHreq, charSegNr;

   // Allocate memory for contour segments
   for (int sg = 0; sg < maxsegment; sg ++)
     {
      Seg[sg] = new Segment;
     }
```

```
if (! SegmentFile)
  {
   MessageBox ("Unable to open file: Segment contours file", "FileError",
               MB_OK | MB_ICONEXCLAMATION);
  }
else
  {
   // reads the header for the segment file
   SegmentFile >> charX >> charY >> charHreq >> charSegNr;
   prevSegNr = -999;
   SegmentFile >> x >> y >> h >> SegNr;
   Transform(MaxY, x, y);
   VecWorld.x = x;
   VecWorld.y = y;
   VecWorld.z = h;

   VectorMatrix(VecWorld, viewT, VecView);

   x = VecView.x;
   y = VecView.y;
   h = VecView.z;

   x = (x - xmin)* scaleX;
   y = (rect.bottom - rect.top) - ((y - ymin) * scaleY);

   Perspective(x, y, h, screen_x, screen_y);

   x = screen_x;
   y = screen_y;

   startx = x;
   starty = y;

   while (! SegmentFile.eof())
     {
      SegmentFile >> nextx >> nexty >> nexth >> nextSegNr;
      Transform(MaxY, nextx, nexty);
      VecWorld.x = nextx;
      VecWorld.y = nexty;
      VecWorld.z = nexth;

      VectorMatrix(VecWorld, viewT, VecView);

      nextx = VecView.x;
      nexty = VecView.y;
      nexth = VecView.z;

      nextx = (nextx - xmin)* scaleX;
      nexty = (rect.bottom - rect.top) - ((nexty - ymin) * scaleY);

      Perspective(nextx, nexty, nexth, screen_x, screen_y);

      nextx = screen_x;
      nexty = screen_y;

      if (SegNr != prevSegNr)
        {
         dc.MoveTo(x, y);
         startx = x;
         starty = y;
        }
```

```
          else
            {
             if ( (x == startx) && (y == starty) )
               {
                if (SegNr == nextSegNr)
                   dc.MoveTo(x, y);
                else
                   dc.LineTo(x, y);
               }
             else
             dc.LineTo(x, y);
            }

          prevSegNr = SegNr;
          SegNr = nextSegNr;

          x = nextx;
          y = nexty;
          h = nexth;
         }

       dc.MoveTo(x, y);
       SegmentFile.close();
      }

      SelectObject(dc, pensegment);
      DeleteObject(pensegment);

    DeallocateMemory();
   }

void TView3D :: CmInputFilesItem1()
   {
    ReadXYZ();
   }

void TView3D :: CmPopupItem1()
   {
    ReadTIN();
   }

// Force the window to repaint if resize
void TView3D :: EvSize(uint, TSize&)
   {
    :: InvalidateRect(HWindow, 0, true);
   }

void TView3D :: CmClear()
   {
    // to clear and redraw
    Invalidate();
   }


void TView3D :: CmAbout()
   {
    MessageBox ("3D TINs display with menu - by Alias Abdul-Rahman, 1999",
                "About the 2D/3D TIN", MB_OK | MB_ICONINFORMATION);
   }
```

```
void TView3DWindowApp :: InitMainWindow()
  {
   // construct the decorated MDI frame window
   TDecoratedFrame* frame = new TDecoratedFrame(0,
                      "3D Viewing (wire-frame)", new TView3D);

   // construct the status bar
   TStatusBar* sb = new TStatusBar(frame, TGadget :: Recessed,
                                     TStatusBar :: CapsLock | TStatusBar ::
NumLock);

   // construct the control bar
   TControlBar* cb = new TControlBar(frame);

   cb -> Insert(*new TButtonGadget(CM_INPUTFILESITEM1, CM_INPUTFILESITEM1,
                    TButtonGadget :: Command));
   cb -> Insert(*new TButtonGadget(CM_POPUPITEM1, CM_POPUPITEM1,
                    TButtonGadget :: Command));

   cb -> Insert(*new TSeparatorGadget);

   cb -> Insert(*new TButtonGadget(CM_CLEAR, CM_CLEAR,
                    TButtonGadget :: Command));

   cb -> Insert(*new TSeparatorGadget);

   cb -> Insert(*new TButtonGadget(CM_FILEEXIT, CM_FILEEXIT,
                    TButtonGadget :: Command));

   cb -> Insert(*new TSeparatorGadget);

   cb -> Insert(*new TButtonGadget(CM_ABOUT, CM_ABOUT,
                    TButtonGadget :: Command));

   cb -> SetHintMode(TGadgetWindow :: EnterHints);

   // set client area to the application workspace
   frame -> SetBkgndColor(::GetSysColor(COLOR_APPWORKSPACE));

   // insert the status and control bar into the frame
   frame -> Insert(*sb, TDecoratedFrame :: Bottom);
   frame -> Insert(*cb, TDecoratedFrame :: Top);

   // set the main window and its menu
   SetMainWindow(frame);
   GetMainWindow() -> AssignMenu("COMMANDS");
   EnableCtl3d(true);
  }


int OwlMain(int /*argc*/, char* /*argv*/ [])
  {
   return TView3DWindowApp().Run();
  }
```

```
//************************************************************************
// Project: view3d
// File: view3d.rh
//
//************************************************************************

#define CM_CLEAR           1125
#define CM_FILEEXIT        1126
#define CM_ABOUT           1127
#define CM_INPUTFILESITEM1    18874 // for XYZ file
#define CM_POPUPITEM1      18873 // for TIN file
```

```
//************************************************************************
// Project: view3D
// File: view3D.rc
//
//************************************************************************


#include "view3d.rh"

COMMANDS MENU
{
 POPUP "&InputFiles"
 {
  MENUITEM "&XYZ File", CM_INPUTFILESITEM1
  MENUITEM "&TIN File", CM_POPUPITEM1
 }

 POPUP "&Redraw"
 {
  MENUITEM "&Clear and Redraw", CM_CLEAR
  MENUITEM "E&xit", CM_FILEEXIT
 }

 MENUITEM "&About", CM_ABOUT
}

STRINGTABLE
  {
   CM_INPUTFILESITEM1,  "Get the XYZ file"
   CM_POPUPITEM1,       "Get the TIN file"
   CM_CLEAR,            "Redraw the TINs"
   CM_FILEEXIT,         "Exit the program ..."
   CM_ABOUT,            "About the program ..."
  }

CM_CLEAR BITMAP
{
 '42 4D 66 01 00 00 00 00 00 00 76 00 00 00 28 00'
 '00 00 14 00 00 00 14 00 00 00 01 00 04 00 00 00'
 '00 00 F0 00 00 00 00 00 00 00 00 00 00 00 00 00'
 '00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 80'
 '00 00 00 80 80 00 80 00 00 00 80 00 80 00 80 80'
 '00 00 C0 C0 C0 00 80 80 80 00 00 00 FF 00 00 FF'
 '00 00 00 FF FF 00 FF 00 00 00 FF 00 FF 00 FF FF'
 '00 00 FF FF FF 00 91 77 77 77 77 77 77 77 77 99'
 '00 00 79 77 77 77 77 77 77 77 79 17 00 00 77 17'
 '77 87 77 77 77 79 91 77 00 00 77 77 80 08 88 87'
 '77 77 77 77 00 00 77 78 0A 00 08 88 77 77 77 77'
 '00 00 77 78 0A 0A A0 08 88 87 77 77 00 00 77 78'
 '0A 0A AA A0 88 88 77 77 00 00 77 80 AA AA AA AA'
 '00 88 77 77 00 00 77 80 AA AA AA A0 A0 88 77 77'
 '00 00 77 0A AA A0 AA 0A A0 88 71 17 00 00 77 0A'
 'AA A0 00 AA A0 88 79 97 00 00 78 0A AA A0 AA AA'
 'A0 88 77 77 00 00 78 AA A0 0A AA AA A0 88 77 77'
 '00 00 78 88 87 78 0A AA A0 88 77 77 00 00 77 77'
 '77 77 80 AA A0 88 77 77 00 00 77 71 77 77 78 0A'
 'A0 88 77 77 00 00 77 19 77 77 77 80 A0 88 77 77'
 '00 00 77 97 77 77 77 78 A0 88 79 77 00 00 71 77'
 '77 77 77 77 88 87 71 19 00 00 19 77 77 77 77 77'
 '77 77 77 71 00 00'
```

```
}


CM_FILEEXIT BITMAP
{
 '42 4D 66 01 00 00 00 00 00 00 76 00 00 00 28 00'
 '00 00 14 00 00 00 14 00 00 00 01 00 04 00 00 00'
 '00 00 F0 00 00 00 00 00 00 00 00 00 00 00 00 00'
 '00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 80'
 '00 00 00 80 80 00 80 00 00 00 80 00 80 00 80 80'
 '00 00 C0 C0 C0 00 80 80 80 00 00 00 FF 00 00 FF'
 '00 00 00 FF FF 00 FF 00 00 00 FF 00 FF 00 FF FF'
 '00 00 FF FF FF 00 EE EE EE EE EE EE EE EE EE EE'
 '00 00 EE EE FF EF EF FE EE EE EE EE 00 00 EE EE'
 'FF EE FE EF EE EE EE EE 00 00 EE EE FF EF FF FF'
 'EE EE EE EE 00 00 EE EE EF EE EF FF EE EE EE EE'
 '00 00 88 88 88 88 88 88 88 88 88 88 00 00 66 66'
 '8A AA A8 77 86 66 66 66 00 00 66 66 8A AA A8 77'
 '86 66 66 66 00 00 66 66 8A AA A8 77 86 66 66 66'
 '00 00 66 66 8A A9 A8 77 86 66 66 66 00 00 66 66'
 '8A A9 A8 77 86 66 66 66 00 00 66 66 8A AA A8 77'
 '86 66 66 66 00 00 66 66 8A AA A8 77 86 66 66 66'
 '00 00 66 66 8A AA A8 77 86 66 66 66 00 00 66 66'
 '8A A8 87 77 86 66 66 66 00 00 66 66 8A 87 77 77'
 '86 66 66 66 00 00 66 66 88 77 77 77 86 66 66 66'
 '00 00 66 66 88 88 88 88 86 66 66 66 00 00 66 66'
 '66 66 66 66 66 66 66 66 00 00 66 66 66 66 66 66'
 '66 66 66 66 00 00'
}


CM_ABOUT BITMAP
{
 '42 4D 66 01 00 00 00 00 00 00 76 00 00 00 28 00'
 '00 00 14 00 00 00 14 00 00 00 01 00 04 00 00 00'
 '00 00 F0 00 00 00 00 00 00 00 00 00 00 00 00 00'
 '00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 80'
 '00 00 00 80 80 00 80 00 00 00 80 00 80 00 80 80'
 '00 00 C0 C0 C0 00 80 80 80 00 00 00 FF 00 00 FF'
 '00 00 00 FF FF 00 FF 00 00 00 FF 00 FF 00 FF FF'
 '00 00 FF FF FF 00 BB BB BB BB BB BB BB BB BB BB'
 '00 00 BB BB BB BB BB BB BB BB BB BB 00 00 BB BB'
 'BB BB BB BB BB BB BB BB 00 00 BB BB BB B8 88 88'
 'BB BB BB BB 00 00 BB BB BB B8 88 88 BB BB BB BB'
 '00 00 BB BB BB B8 00 88 BB BB BB BB 00 00 BB BB'
 'BB B8 00 BB BB BB BB BB 00 00 BB BB BB BB BB BB'
 'BB BB BB BB 00 00 BB BB BB B8 00 88 BB BB BB BB'
 '00 00 BB BB BB BB 80 08 BB BB BB BB 00 00 BB BB'
 'B8 8B B8 08 8B BB BB BB 00 00 BB BB B8 80 BB 00'
 'BB BB BB BB 00 00 BB BB B8 80 BB 00 8B BB BB BB'
 '00 00 BB BB BB 88 88 88 BB BB BB BB 00 00 BB BB'
 'BB B8 88 88 8B BB BB BB 00 00 BB BB BB B8 88 88'
 'BB BB BB BB 00 00 BB BB BB BB BB BB BB BB BB BB'
 '00 00 BB BB BB BB BB BB BB BB BB BB 00 00 BB BB'
 'BB BB BB BB BB BB BB BB 00 00 BB BB BB BB BB BB'
 'BB BB BB BB 00 00'
}

CM_INPUTFILESITEM1 BITMAP
{
 '42 4D 66 01 00 00 00 00 00 00 76 00 00 00 28 00'
```

```
 '00 00 14 00 00 00 14 00 00 00 01 00 04 00 00 00'
 '00 00 F0 00 00 00 00 00 00 00 00 00 00 00 00 00'
 '00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 80'
 '00 00 00 80 80 00 80 00 00 00 80 00 80 00 80 80'
 '00 00 C0 C0 C0 00 80 80 80 00 00 00 FF 00 00 FF'
 '00 00 00 FF FF 00 FF 00 00 00 FF 00 FF 00 FF FF'
 '00 00 FF FF FF 00 77 77 77 77 77 77 77 77 77 77'
 '00 00 77 00 00 00 00 00 00 00 08 77 00 00 77 0A'
 'AA AA AA AA AA AA 08 77 00 00 77 0A 99 9A AA AA'
 'AA AA 08 77 00 00 77 0A AA AA AA AA AA AA 08 77'
 '00 00 77 0A AA AA AA AA AA AA 08 77 00 00 77 00'
 '00 00 00 00 00 00 08 77 00 00 77 88 88 88 88 88'
 '88 88 88 77 00 00 77 77 77 77 77 77 77 77 77 77'
 '00 00 77 00 00 00 00 00 00 77 77 77 00 00 77 0F'
 'FF FF FF FF F0 77 77 77 00 00 77 0F 00 0F FF FF'
 'F0 77 77 77 00 00 77 0F FF FF F0 F0 F0 77 77 77'
 '00 00 77 0F 0F 0F 0F 0F F0 77 77 77 00 00 77 0F'
 'FF FF FF FF 08 77 77 77 00 00 77 0F 00 0F 0F FF'
 '88 77 77 77 00 00 77 0F FF FF FF 00 88 77 77 77'
 '00 00 77 00 00 00 00 88 88 77 77 77 00 00 77 77'
 '77 77 77 77 77 77 77 77 00 00 77 77 77 77 77 77'
 '77 77 77 77 00 00'
}


CM_POPUPITEM1 BITMAP
{
 '42 4D 66 01 00 00 00 00 00 00 76 00 00 00 28 00'
 '00 00 14 00 00 00 14 00 00 00 01 00 04 00 00 00'
 '00 00 F0 00 00 00 00 00 00 00 00 00 00 00 00 00'
 '00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 80'
 '00 00 00 80 80 00 80 00 00 00 80 00 80 00 80 80'
 '00 00 C0 C0 C0 00 80 80 80 00 00 00 FF 00 00 FF'
 '00 00 00 FF FF 00 FF 00 00 00 FF 00 FF 00 FF FF'
 '00 00 FF FF FF 00 77 77 77 77 77 77 77 77 77 77'
 '00 00 77 00 00 00 00 00 00 00 08 77 00 00 77 0A'
 'AA AA AA AA AA AA 08 77 00 00 77 0A 99 9A AA AA'
 'AA AA 08 77 00 00 77 0A AA AA AA AA AA AA 08 77'
 '00 00 77 0A AA AA AA AA AA AA 08 77 00 00 77 00'
 '00 00 00 00 00 00 08 77 00 00 77 88 88 88 88 88'
 '88 88 88 77 00 00 77 77 77 77 77 77 77 77 77 77'
 '00 00 77 70 00 00 00 00 00 07 77 77 00 00 77 70'
 'FF FF FF FF FF 07 77 77 00 00 77 70 F0 FF FF 00'
 'FF 07 77 77 00 00 77 70 FF FF 00 FF FF 07 77 77'
 '00 00 77 70 F0 0F FF FF FF 07 77 77 00 00 77 70'
 'FF FF F0 0F FF 07 77 77 00 00 77 70 FF FF FF FF'
 'F0 87 77 77 00 00 77 70 FF 00 0F FF 08 87 77 77'
 '00 00 77 70 FF FF FF F0 88 87 77 77 00 00 77 70'
 '00 00 00 88 88 87 77 77 00 00 77 77 77 77 77 77'
 '77 77 77 77 00 00'
}
```

POET Database Compatible Code

```
//************************************************************//
//  Project: TIN-based Spatial Modelling using POET           //
//  File: main.cpp                                            //
//  Author: Alias Abdul-Rahman (c) Feb., 2000                 //
//                                                            //
//************************************************************//

#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include <fstream.h>
#include <wtypes.h>
#include <conio.h>

// POET include file
#include <poet.hxx>

// PTXX generated file
#include "tnode.cxx"
#include "tedge.cxx"
#include "tpoly.cxx"
#include "tsolid.cxx"


int main(int argc, char *argv[])
  {
   // Initiliase the POET database
   InitPOET();
   PtBase* pObjBase;
   int err = PtBase :: POET()->GetBase("LOCAL", "base", pObjBase);
   if (err < 0)
     {
      PtBase :: POET()->UngetBase(pObjBase);
      DeinitPOET();
      return err;
     }

   // to handle the TNode class
   TNode Node;
   Node.GetXYZCoordinates();
   Node.GetBoreholeCoordinates();
   Node.Get2Node();
   Node.NodeAttribute();

   // to store the Nodes objects in the database
   Node.Assign(pObjBase);
   err = Node.Store();

   // to handle the TEdge class
   TEdge Edge;
   Edge.ReadARCs();
   Edge.GetOneArc();
```

```
  Edge.EdgeAttribute();
  Edge.GetArcLength();

  // to stote the Edge objects in the database
  Edge.Assign(pObjBase);
  err = Edge.Store();

  // to handle the TPoly class
  TPoly Poly;
  Poly.ReadTINs();
  Poly.GetTINNeighbour();
  Poly.GetPolyArea();

  // to stote the TPoly object in the database
  Poly.Assign(pObjBase);
  err = Poly.Store();

  // to handle the TSolid class
  TSolid Solid;
  Solid.ReadTENs();
  Solid.GetVolume();

  // to store the Solid objects in the database
  Solid.Assign(pObjBase);
  err = Solid.Store();

  // to close the database
  PtBase :: POET()->UngetBase(pObjBase);
  DeinitPOET();
 }
```

```
//************************************************************//
//  Project: TIN-based Spatial Modelling using POET           //
//  File: tnode.cpp                                           //
//  Author: Alias Abdul-Rahman (c) Feb., 2000                 //
//                                                            //
//************************************************************//

#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include <fstream.h>
#include <string>
#include <conio.h>


// POET compiler includes
#include <poet.hxx>

// PTXX generated files
#include "tnode.hxx"

#include "xyzcontainer.hxx"
#include "nodenamecontainer.hxx"

// Borland C++ includes
//#include "tnode.h"
//#include "xyzcontainer.h"
//#include "nodenamecontainer.h"


#define maxpoint 10000
#define maxnodename 100

// forward declaration
XYZContainer Point[maxpoint];
NodeNameContainer NodeAtr[maxnodename];
//NodeAtrContainer NodeAtr[maxnodename];

// implementation of class TNode
// the constructor
TNode :: TNode()
  {

  }

// the destructor
TNode :: ~TNode()
  {

  }


void TNode :: PressAnyKey()
  {
   cout << endl;
   cout << endl;
   cout << "Press any key to continue" << endl;
   getch();
   cout << endl;
  }
```

```
void TNode :: GetXYZCoordinates()
  {
   string charX, charY, charZ;

   // to read the XYZ file
   ifstream XYZfile("c:\\data\\lochass.xyz");
   if (! XYZfile)
     cout << "Could not open the file" << endl;
   else
     {
      // reads the file header
      XYZfile >> charX >> charY >> charZ;
      int npnt = 0;
      do
        {
         npnt ++;
         printf("\r");
         cout << "reading points ...." << npnt;
         XYZfile >> Point[npnt].x >> Point[npnt].y >> Point[npnt].z;
        } while (! XYZfile.eof());
     }

   cout << endl;
   cout << endl;
   XYZfile.close();
  }

void TNode :: GetBoreholeCoordinates()
  {
   string charX, charY, charZ;

   // to read the XYZ file
   ifstream XYZBorefile("c:\\data\\lakebore.xyz");
   if (! XYZBorefile)
     cout << "Could not open the file" << endl;
   else
     {
      // reads the file header
      XYZBorefile >> charX >> charY >> charZ;
      int npnt = 0;
      do
        {
         npnt ++;
         printf("\r");
         cout << "reading points ...." << npnt;
         XYZBorefile >> Point[npnt].x >> Point[npnt].y >> Point[npnt].z;
        } while (! XYZBorefile.eof());
     }

   cout << endl;
   cout << endl;
   XYZBorefile.close();
  }


void TNode :: Get2Node()
  {
   int selectnode1, selectnode2;
   int npnt1, npnt2;
   double Node1x, Node1y;
   double Node2x, Node2y;
```

```
   cout << "The point# range: 1 to " << npnt << endl;
   cout << "Select node 1 :" << endl;
   cin >> selectnode1;
   cout << "Select node 2:" << endl;
   cin >> selectnode2;
   npnt1 = selectnode1;
   npnt2 = selectnode2;
   Node1x = Point[npnt1].x;
   Node1y = Point[npnt1].y;
   Node2x = Point[npnt2].x;
   Node2y = Point[npnt2].y;
   cout << endl;
   cout << "The selected points are:" << endl;
   cout << "    Node1  : " << npnt1 << endl;
   cout << "     (x, y): (" << Node1x << ", " << Node1y << ")" << endl;
   cout << "    Node2  : " << npnt2 << endl;
   cout << "     (x, y): (" << Node2x << ", " << Node2y << ")" << endl;
   cout << endl;
  }

void TNode :: NodeAttribute()
  {
   string charNodeNr, charNodeAtr;

   // reads the node attribute
   ifstream NodeAtrFile("c:\\data\\node.atr");

   if (! NodeAtrFile)
     cout << "Could not open the Node attribute file" << endl;
   else
     {
      // reads the file header
      NodeAtrFile >> charNodeNr >> charNodeAtr;

      int nnode = 0;
      do
        {
         nnode ++;
         printf("\r");
         cout << "reading the node attribute: " << nnode;
         NodeAtrFile >> NodeAtr[nnode].NodeNum >> NodeAtr[nnode].NodeName;
        } while ( ! NodeAtrFile.eof());
     }
   cout << endl;

  }
```

```
//************************************************************//
//  Project: TIN-based Spatial Modelling using POET           //
//  File: tedge.cpp                                           //
//  Author: Alias Abdul-Rahman (c) Feb., 2000                 //
//                                                            //
//************************************************************//

#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include <string>
#include <fstream.h>
#include <math.h>
#include <conio.h>

// POET compiler includes
#include <poet.hxx>

// PTXX generated files
#include "tedge.hxx"
#include "edgenamecontainer.hxx"
#include "tnode.hxx"

// Borland C++ includes
//#include "tedge.h"
//#include "edgenamecontainer.h"
//#include "arccontainer.h"


#define maxpoint 10000
#define maxarc 200
#define accy 1e-10


// forward declaration
TNode Node;
ARCContainer Arcs[maxarc];
EdgeNameContainer EdgeAtr[maxarc];
XYZContainer Point[maxpoint];

// the implementation for the TEdge class
TEdge :: TEdge()
  {

  }

TEdge :: ~TEdge()
  {

  }

void TEdge :: PressAnyKey()
  {
   cout << endl;
   cout << endl;
   cout << "Press any key to continue" << endl;
   getch();
   cout << endl;
  }

void TEdge :: ReadARCs()
```

```
  {
   string charStartNode, charEndNode;

   // reads the arc file
   ifstream ARCfile = "c:\\data\\lake.arc";

   if (!ARCfile)
     cout << "Unable to open file: ARC file" << endl;

   else
     {
      ARCfile >> charStartNode >> charEndNode;
      int narc = 0;
      do
        {
         narc ++;
         printf("\r");
         cout << "reading ARCs ...." << narc;
         ARCfile >> Arcs[narc].StartNode
                 >> Arcs[narc].EndNode;
        } while (! ARCfile.eof());
     }
   cout << endl;
   cout << endl;
   ARCfile.close();
  }


void TEdge :: GetOneArc()
  {
   int arcnum;
   int Node1, Node2;
   cout << "Enter one arc #: " << endl;
   cin >> arcnum;
   Node1 = Arcs[arcnum].StartNode;
   Node2 = Arcs[arcnum].EndNode;
   cout << "StartNode : " << Node1 << endl;
   cout << "EndNode   : " << Node2 << endl;
   //PressAnyKey();
  }

void TEdge :: GetArcLength()
  {
   int arcno;
   int ArcStartNode, ArcEndNode;
   double ArcX1, ArcY1, ArcX2, ArcY2;
   double dist;

   Node.GetXYZCoordinates();
   cout << "Enter the arc #: " << endl;
   cin >> arcno;
   ArcStartNode = Arcs[arcno].StartNode;
   ArcEndNode = Arcs[arcno].EndNode;
   cout << "StarNode: " << ArcStartNode << endl;
   cout << "EndNode : " << ArcEndNode << endl;
   ArcX1 = Node.Point[ArcStartNode].x;
   ArcY1 = Node.Point[ArcStartNode].y;
   ArcX2 = Node.Point[ArcEndNode].x;
   ArcY2 = Node.Point[ArcEndNode].y;

   dist = sqrt((ArcX2-ArcX1)*(ArcX2-ArcX1) + (ArcY2-ArcY1)*(ArcY2-ArcY1));
```

```
   cout << "Node1(x): " << ArcX1 << endl;
   cout << "Node1(y): " << ArcY1 << endl;
   cout << "The arc length: " << dist << endl;
   //PressAnyKey();
  }

int TEdge :: CheckQuadrant(float dx, float dy)
  {
   int c;

   if ((dx > 0) && (dx > 0))
     c = 1;
   if ((dx < 0) && (dy > 0))
     c = 2;
   if ((dx < 0) && (dy < 0))
     c = 3;
   if ((dx > 0) && (dy < 0))
     c = 4;
   return c;
  }

float TEdge :: Bearing(float x1, float y1, float x2, float y2)
  {
   float theta;
   float dx, dy;

   dx = x2 - x1;
   dy = y2 - y2;
   if (abs(dy) < accy)
      {
       theta = M_PI/2;
       return theta;
      }
   else
      return theta = atan(dx/dy);

  }

float TEdge :: GetArcAzimuth(float x1, float y1, float x2, float y2)
  {
   float dx, dy;
   float theta;
   int quadrant;

   dx = x2 - x1;
   dy = y2 - y1;
   if ((dx == 0) && (dy > 0))
     theta = 0;
   else if ((dx > 0) && (dy == 0))
            theta = M_PI/2;
   else if ((dx == 0) && (dy < 0))
            theta = M_PI;
   else if ((dx < 0) && (dy == 0))
            theta = 3 * M_PI/2;
   else
      {
       quadrant = CheckQuadrant(dx, dy);
       switch (quadrant)
         {
          case 1: theta = Bearing(x1, y1, x2, y2);
```

```
          break;

          case 2: theta = (2*M_PI) - abs(Bearing(x1, y1, x2, y2));
          break;

          case 3: theta = M_PI + abs(Bearing(x1, y1, x2, y2));
          break;

          case 4: theta = M_PI - abs(Bearing(x1, y1, x2, y2));
          break;
         }
      }

   return theta;
  }

void TEdge :: EdgeAttribute()
  {
   string charEdgeNr, charEdgeName;

   // reads the Edge attribute file
   ifstream EdgeAtrFile("c:\\data\\edge.atr");

   if (! EdgeAtrFile)
     cout << "Could not open the Edge attribute file" << endl;
   else
     {
      // reads the file header
      EdgeAtrFile >> charEdgeNr >> charEdgeName;

      int nedge = 0;
      do
        {
         nedge ++;
         printf("\r");
         cout << "reading the Edge attribute: " << nedge;
         EdgeAtrFile >> EdgeAtr[nedge].EdgeNum >> EdgeAtr[nedge].EdgeName;
        } while ( ! EdgeAtrFile.eof());
     }
   cout << endl;
   printf("%d   %s\n" , EdgeAtr[5].EdgeNum, EdgeAtr[5].EdgeName);
   //PressAnyKey();
  }
```

```
//************************************************************//
//  Project: TIN-based Spatial Modelling using POET           //
//  File   : tpoly.cpp                                        //
//  Author : Alias Abdul-Rahman (c) Feb., 2000                //
//                                                            //
//************************************************************//

#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include <fstream.h>
#include <string>
#include <conio.h>

// POET includes
#include <poet.hxx>

// PTXX generated files
//#include "tpoly.hxx"

// Borland C++ include
//#include "tpoly.h"
//#include "tnode.h"


//JVG
#include "tpoly.hxx"
#include "tnode.hxx"
#include "tedge.hxx"

#define maxtriangle 10000
#define maxpoint 10000

// forward declaration
TNode Node;
TEdge Edge;

// Implementation for the class TPoly class
TPoly :: TPoly()    // constructor
  {

  }

TPoly :: ~TPoly()   // destructor
  {

  }


void TPoly :: PressAnyKey()
  {
   cout << endl;
   cout << endl;
   cout << "Press any key to continue" << endl;
   getch();
   cout << endl;
  }

void TPoly :: ReadTINs()
  {
   string charNode1, charNode2, charNode3;
```

```
   // reads the TIN file
   ifstream TINfile = "c:\\data\\lochas.tin";

   if (!TINfile)
     cout << "Unable to open file: TIN file " << endl;

   else
     {
      TINfile >> charNode1 >> charNode2 >> charNode3;
      int ntri = 0;
      do
        {
         ntri ++;
         printf("\r");
         cout << "reading TINs ..... " << ntri;
         TINfile >> Triangle[ntri].Node1
                 >> Triangle[ntri].Node2
                 >> Triangle[ntri].Node3;
        } while (! TINfile.eof());
     }

   cout << endl;
   cout << endl;
   TINfile.close();
  }

void TPoly :: GetTINNeighbour()
  {
   string charTriNum, charNumofNbr, charNbr1, charNbr2, charNbr3;

   // to read the TIN neighbour file
   ifstream TINNbrfile("c:\\data\\lochgrd.nbr");
   if (! TINNbrfile)
     cout << "Could not open the TIN neighbour file" << endl;
   else
     {
      // reads the file header
       TINNbrfile >> charTriNum >> charNumofNbr >> charNbr1 >> charNbr2 >>
charNbr3;
      int ntri = 0;
      do
        {
         ntri ++;
         printf("\r");
         cout << "reading TIN Neighbour ...." << ntri;
         TINNbrfile >> TINNbr[ntri].TriNum
                    >> TINNbr[ntri].NumofNbr
                    >> TINNbr[ntri].Nbr1
                    >> TINNbr[ntri].Nbr2
                    >> TINNbr[ntri].Nbr3;
        } while (! TINNbrfile.eof());
     }

   cout << endl;
   cout << endl;
   TINNbrfile.close();
  }

float TPoly :: GetTINArea(double x1, double y1,
                          double x2, double y2,
                          double x3, double y3)
```

```
  {
   double x12, y32;
   double x32, y12;

   float area;

   x12 = x1 - x2;
   x32 = x3 - x2;
   y12 = y1 - y2;
   y32 = y3 - y2;

   area = 0.5 * (x12*y32 - x32*y12);
   return area;
  }

void TPoly :: GetTINNodes(int t,
                          double& xNd1, double& yNd1, double& zNd1,
                          double& xNd2, double& yNd2, double& zNd2,
                          double& xNd3, double& yNd3, double& zNd3)
  {
   int Nd1, Nd2, Nd3;

   Nd1 = Triangle[t].Node1;
   Nd2 = Triangle[t].Node2;
   Nd3 = Triangle[t].Node3;

   xNd1 = Node.Point[Nd1].x;
   yNd1 = Node.Point[Nd1].y;
   zNd1 = Node.Point[Nd1].z;

   xNd2 = Node.Point[Nd2].x;
   yNd2 = Node.Point[Nd2].y;
   zNd2 = Node.Point[Nd2].z;

   xNd3 = Node.Point[Nd3].x;
   yNd3 = Node.Point[Nd3].y;
   zNd3 = Node.Point[Nd3].z;
  }


void TPoly :: GetPolyArea()
  {

   double AreaOneTri;
   double TotalArea = 0;
   double xNd1, yNd1, zNd1;
   double xNd2, yNd2, zNd2;
   double xNd3, yNd3, zNd3;
   double PolyArea;
   int StartTri, LastTri;

   cout << "Enter Begin Triangle #: " << endl;
   cin >> StartTri;
   cout << "Enter End Triangle #: " << endl;
   cin >> LastTri;

   Node.GetXYZCoordinates();

   for (int t = StartTri; t <= LastTri; t ++)
     {
      printf("\r");
```

```
      cout << "calculating area for TIN #: " << t;
      GetTINNodes(t, xNd1, yNd1, zNd1,
                     xNd2, yNd2, zNd2,
                     xNd3, yNd3, zNd3);
      AreaOneTri = GetTINArea(xNd1, yNd1, xNd2, yNd2, xNd3, yNd3);
      cout << endl;
      cout << "Area of Triangle: " << AreaOneTri << endl;
      PolyArea = PolyArea + AreaOneTri;
     }

   cout << endl;
   cout << "Total area of TINs: " << PolyArea << endl;
   //PressAnyKey();
  }

//*************************************************************//
//  Project: TIN-based Spatial Modelling using POET            //
//  File   : tsolid.cpp                                        //
//  Author : Alias Abdul-Rahman (c) Feb., 2000                 //
//                                                             //
//*************************************************************//

#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include <string>
#include <fstream.h>
#include <conio.h>

// POET includes
#include <poet.hxx>

// PTXX generated files
#include "tsolid.hxx"
#include "tnode.hxx"
#include "tpoly.hxx"

// Borland C++ includes
//#include "tsolid.h"
//#include "tpoly.h"
//#include "tnode.h"

#define maxpoint 10000
#define maxtriangle 10000


TNode Node;
TPoly Poly;

// the Implementation for the TSolid class
TSolid :: TSolid()       // constructor
  {

  }

TSolid :: ~TSolid()      // destructor
  {

  }
```

```
void TSolid :: PressAnyKey()
  {
   cout << endl;
   cout << endl;
   cout << "Press any key to continue" << endl;
   getch();
   cout << endl;
  }

void TSolid :: ReadTENs()
  {
   string charNode1, charNode2, charNode3, charNode4;

   // Reads the 3D TIN file
   ifstream TENfile = "c:\\data\\lakebo.ten";

   if (!TENfile)
     cout << "Unable to open file: TEN file" << endl;

   else
     {
      TENfile >> charNode1 >> charNode2 >> charNode3 >> charNode4;
      int nten = 0;
      do
        {
         nten ++;
         printf("\r");
         cout << "reading TENs ... " << nten;
         TENfile >> TEN[nten].Node1
                 >> TEN[nten].Node2
                 >> TEN[nten].Node3
                 >> TEN[nten].Node4;
        } while (!TENfile.eof());
     }

   cout << endl;
   cout << endl;
   TENfile.close();
  }


void TSolid :: Get3TINNodes(int t,
                            double& xNd1, double& yNd1, double& zNd1,
                            double& xNd2, double& yNd2, double& zNd2,
                            double& xNd3, double& yNd3, double& zNd3,
                            double& xNd4, double& yNd4, double& zNd4)
  {
   int Nd1, Nd2, Nd3, Nd4;

   Nd1 = TEN[t].Node1;
   Nd2 = TEN[t].Node2;
   Nd3 = TEN[t].Node3;
   Nd4 = TEN[t].Node4;

   xNd1 = Node.Point[Nd1].x;
   yNd1 = Node.Point[Nd1].y;
   zNd1 = Node.Point[Nd1].z;

   xNd2 = Node.Point[Nd2].x;
   yNd2 = Node.Point[Nd2].y;
   zNd2 = Node.Point[Nd2].z;
```

```
   xNd3 = Node.Point[Nd3].x;
   yNd3 = Node.Point[Nd3].y;
   zNd3 = Node.Point[Nd3].z;

   xNd4 = Node.Point[Nd4].x;
   yNd4 = Node.Point[Nd4].y;
   zNd4 = Node.Point[Nd4].z;
  }

double TSolid :: ComputeTENVolume(double x1, double y1, double z1,
                                  double x2, double y2, double z2,
                                  double x3, double y3, double z3,
                                  double x4, double y4, double z4)
  {
   double x41, y41, z41;
   double x31, y31, z31;
   double x21, y21, z21;
   double d32, d42, d43;

   x41 = x4 - x1;
   y41 = y4 - y1;
   z41 = z4 - z1;

   x31 = x3 - x1;
   y31 = y3 - y1;
   z31 = z3 - z1;

   x21 = x2 - x1;
   y21 = y2 - y1;
   z21 = z2 - z1;

   d32 = y31*z21 - z31*y21;
   d42 = y41*z21 - z41*y21;
   d43 = y41*z31 - z41*y31;

   return abs(0.166667 * (x41*d32 - x31*d42 + x21*d43));
  }

void TSolid :: GetVolume()
  {
   double VolOneTEN;
   double TotalVolume;
   int TEN1, TEN2;

   Node.GetBoreholeCoordinates();
   cout << "Enter range of TEN #: " << endl;
   cout << "         first TEN #: " << endl;
   cin >> TEN1;
   cout << "         last  TEN #: " << endl;
   cin >> TEN2;

   for (int t = TEN1; t <= TEN2; t ++)
     {
      Get3TINNodes(t, xNd1, yNd1, zNd1,
                      xNd2, yNd2, zNd2,
                      xNd3, yNd3, zNd3,
                      xNd4, yNd4, zNd4);

      VolOneTEN = ComputeTENVolume(xNd1, yNd1, zNd1,
                                   xNd2, yNd2, zNd2,
                                   xNd3, yNd3, zNd3,
```

```
                                    xNd4, yNd4, zNd4);
     TotalVolume = TotalVolume + VolOneTEN;
    }
 cout << endl;
 cout << "Total volume of the given TENs: " << TotalVolume << endl;
 //PressAnyKey();
}
```

User Interface (MDI) Code

```
//****************************************************************//
//  Project: User Interface (MDI)for 2D/3D TINs
//
//  SUBSYSTEM:    Graphing Application
//
//  FILE:         graphingapp.h
//
//
//
//  OVERVIEW
//
//  ~~~~~~~~
//
//  Class definition for GraphingApp (TApplication).
//
//  Author: Alias Abdul-Rahman (c) 1999
//
//****************************************************************//

#if !defined(graphingapp_h)        // Sentry, use file only if it's not
#define graphingapp_h              //   already included.

#include <owl/controlb.h>
#include <owl/docking.h>
#include <owl/printer.h>
#include <owl/rcntfile.h>

#include "graphingmdiclient.h"

#include "graphingapp.rh"          // Definition of all resources.


//{{TApplication = GraphingApp}}
class GraphingApp : public TApplication, public TRecentFiles {
  private:
    void SetupSpeedBar(TDecoratedMDIFrame* frame);

  public:
    GraphingApp();
    virtual ~GraphingApp();

    void CreateGadgets(TDockableControlBar* cb, bool server = false);
    THarbor*        ApxHarbor;

    TGraphingMDIClient* MdiClient;

    // Public data members used by the print menu commands
    // and Paint routine in MDIChild.
    //
    TPrinter*       Printer;               // Printer support.
    int             Printing;              // Printing in progress.
```

```
//{{GraphingAppVIRTUAL_BEGIN}}
  public:
    virtual void InitMainWindow();
//{{GraphingAppVIRTUAL_END}}

//{{GraphingAppRSP_TBL_BEGIN}}
  protected:
    void EvNewView(TView& view);
    void EvCloseView(TView& view);
    void CmHelpAbout();
    void EvWinIniChange(char far* section);
    void EvOwlDocument(TDocument& doc);
    int32 CmFileSelected(uint wp, int32 lp);
//{{GraphingAppRSP_TBL_END}}
DECLARE_RESPONSE_TABLE(GraphingApp);
};    //{{GraphingApp}}


#endif  // graphingapp_h sentry.
```

```
//************************************************************************//
//  Project: User Interface (MDI) for 2D/3D TIN
//
//  SUBSYSTEM:    Graphing Application
//
//  FILE:         graphingapp.cpp
//
//
//  OVERVIEW
//
//  Source file for implementation of TGraphingApp.
//
//
//
//  Author: Alias Abdul-Rahman (c) 1999
//
//************************************************************************//

#include <owl/pch.h>

#include <owl/buttonga.h>
#include <owl/statusba.h>
#include <owl/docmanag.h>
#include <owl/filedoc.h>

#include "graphingapp.h"
#include "graphingmdiclient.h"
#include "graphingmdichild.h"
#include "graphingwindowview.h"
#include "graphingaboutdlg.h"             // Definition of about dialog.
#include "graphdocument.h"                // Definition of about dialog.

//{{GraphingApp Implementation}}



//{{DOC_VIEW}}
DEFINE_DOC_TEMPLATE_CLASS(GraphDocument, TGraphingWindowView, DocType1);

//{{DOC_VIEW_END}}

//{{DOC_MANAGER}}
DocType1 __dvt1("TIN Files", "*.xyz", 0, "*.xyz", dtAutoDelete | dtUpdateDir
                                                                        |
dtOverwritePrompt);

//{{DOC_MANAGER_END}}


//
//  Build  a  response  table  for  all  messages/commands  handled  by  the
application.
//
DEFINE_RESPONSE_TABLE2(GraphingApp, TRecentFiles, TApplication)
//{{GraphingAppRSP_TBL_BEGIN}}
  EV_OWLVIEW(dnCreate, EvNewView),
  EV_OWLVIEW(dnClose,  EvCloseView),
  EV_COMMAND(CM_HELPABOUT, CmHelpAbout),
  EV_WM_WININICHANGE,
  EV_OWLDOCUMENT(dnCreate, EvOwlDocument),
```

```
  EV_OWLDOCUMENT(dnRename, EvOwlDocument),
  EV_REGISTERED(MruFileMessage, CmFileSelected),
//{{GraphingAppRSP_TBL_END}}
END_RESPONSE_TABLE;


//----------------------------------------------------------
// GraphingApp
// ~~~~~
//
GraphingApp::GraphingApp() : TApplication("TINSoft ver. 1.0"),
                                          TRecentFiles(".\\Graphing.ini", 4)
{
  Printer = 0;
  Printing = 0;

  SetDocManager(new TDocManager(dmMDI, this));

  // INSERT>> Your constructor code here.
}


GraphingApp::~GraphingApp()
{
  delete Printer;

  // INSERT>> Your destructor code here.
}


void GraphingApp::CreateGadgets(TDockableControlBar* cb, bool server)
{
  if (!server) {
    cb->Insert(*new TButtonGadget(CM_MDIFILENEW, CM_MDIFILENEW));
    cb->Insert(*new TButtonGadget(CM_MDIFILEOPEN, CM_MDIFILEOPEN));
    cb->Insert(*new TButtonGadget(CM_FILESAVE, CM_FILESAVE));
    cb->Insert(*new TSeparatorGadget(6));
  }

  //cb->Insert(*new TButtonGadget(CM_EDITCUT, CM_EDITCUT));
  //cb->Insert(*new TButtonGadget(CM_EDITCOPY, CM_EDITCOPY));
  //cb->Insert(*new TButtonGadget(CM_EDITPASTE, CM_EDITPASTE));
  //cb->Insert(*new TSeparatorGadget(6));
  //cb->Insert(*new TButtonGadget(CM_EDITUNDO, CM_EDITUNDO));
  //cb->Insert(*new TSeparatorGadget(6));
  //cb->Insert(*new TButtonGadget(CM_EDITFIND, CM_EDITFIND));
  //cb->Insert(*new TButtonGadget(CM_EDITFINDNEXT, CM_EDITFINDNEXT));

  if (!server) {
    cb->Insert(*new TSeparatorGadget(6));
    cb->Insert(*new TButtonGadget(CM_FILEPRINT, CM_FILEPRINT));
   cb->Insert(*new TButtonGadget(CM_FILEPRINTPREVIEW, CM_FILEPRINTPREVIEW));
  }

  // Add caption and fly-over help hints.
  //
  cb->SetCaption("Toolbar");
  cb->SetHintMode(TGadgetWindow::EnterHints);
}
```

```
void GraphingApp::SetupSpeedBar(TDecoratedMDIFrame* frame)
{
  ApxHarbor = new THarbor(*frame);

  // Create default toolbar New and associate toolbar buttons with commands.
  //
  TDockableControlBar* cb = new TDockableControlBar(frame);
  CreateGadgets(cb);

  // Setup the toolbar ID used by OLE 2 for toolbar negotiation.
  //
  cb->Attr.Id = IDW_TOOLBAR;

  ApxHarbor->Insert(*cb, alTop);
}


//----------------------------------------------------------
// GraphingApp
// ~~~~~
// Application main window construction & intialization.
//
void GraphingApp::InitMainWindow()
{
  if (nCmdShow != SW_HIDE)
    nCmdShow = (nCmdShow != SW_SHOWMINNOACTIVE) ? SW_SHOWNORMAL : nCmdShow;

  MdiClient = new TGraphingMDIClient(this);
  TDecoratedMDIFrame* frame = new TDecoratedMDIFrame(Name, IDM_MDI,
                                                     *MdiClient, true, this);

  nCmdShow = (nCmdShow != SW_SHOWMINNOACTIVE) ? SW_SHOWNORMAL : nCmdShow;

  // Assign icons for this application.
  //
  frame->SetIcon(this, IDI_MDIAPPLICATION);
  frame->SetIconSm(this, IDI_MDIAPPLICATION);

  // Associate with the accelerator table.
  //
  frame->Attr.AccelTable = IDM_MDI;

  TStatusBar* sb = new TStatusBar(frame, TGadget::Recessed,
                                  TStatusBar::CapsLock        |
                                  TStatusBar::NumLock         |
                                  TStatusBar::ScrollLock);
  frame->Insert(*sb, TDecoratedFrame::Bottom);

  SetupSpeedBar(frame);

  SetMainWindow(frame);

  frame->SetMenuDescr(TMenuDescr(IDM_MDI));

}



//----------------------------------------------------------
// GraphingApp
// ~~~~~
```

```
// Response Table handlers:
//
void GraphingApp::EvNewView(TView& view)
{
          T M D I C l i e n t *      m d i C l i e n t      =
TYPESAFE_DOWNCAST(GetMainWindow()->GetClientWindow(),
                                                          TMDIClient);
  if (mdiClient) {
    TGraphingMDIChild* child = new TGraphingMDIChild(*mdiClient, 0,
                                                     view.GetWindow());

    // Set the menu descriptor for this view.
    //
    if (view.GetViewMenu())
      child->SetMenuDescr(*view.GetViewMenu());

    // Assign icons with this child window.
    //
    child->SetIcon(this, IDI_DOC);
    child->SetIconSm(this, IDI_DOC);

    child->Create();
  }
}


void GraphingApp::EvCloseView(TView&)
{
}


//----------------------------------------------------------
// GraphingApp
// ~~~~~~~~~~~
// Menu Help About Graphing command
void GraphingApp::CmHelpAbout()
{
  // Show the modal dialog.
  //
  GraphingAboutDlg(GetMainWindow()).Execute();
}


void GraphingApp::EvOwlDocument(TDocument& doc)
{
  if (doc.GetDocPath())
    SaveMenuChoice(doc.GetDocPath());
}


int32 GraphingApp::CmFileSelected(uint wp, int32)
{
  TAPointer<char> text = new char[_MAX_PATH];

  GetMenuText(wp, text, _MAX_PATH);
  TDocTemplate* tpl = GetDocManager()->MatchTemplate(text);
  if (tpl)
    GetDocManager()->CreateDoc(tpl, text);
  return 0;
}
```

```
void GraphingApp::EvWinIniChange(char far* section)
{
  if (strcmp(section, "windows") == 0) {
    // If the device changed in the WIN.INI file then the printer
    // might have changed.  If we have a TPrinter(Printer) then
    // check and make sure it's identical to the current device
    // entry in WIN.INI.
    //
    if (Printer) {
      const int bufferSize = 255;
      char printDBuffer[bufferSize];
      LPSTR printDevice = printDBuffer;
      LPSTR devName;
      LPSTR driverName = 0;
      LPSTR outputName = 0;
           if  (::GetProfileString("windows",  "device",  "",  printDevice,
bufferSize)) {
        // The string which should come back is something like:
        //
        //      HP LaserJet III,hppcl5a,LPT1:
        //
        // Where the format is:
        //
        //      devName,driverName,outputName
        //
        devName = printDevice;
        while (*printDevice) {
          if (*printDevice == ',') {
            *printDevice++ = 0;
            if (!driverName)
              driverName = printDevice;
            else
              outputName = printDevice;
          }
          else
            printDevice = ::AnsiNext(printDevice);
        }

        if (Printer->GetSetup().Error != 0                 ||
            strcmp(devName, Printer->GetSetup().GetDeviceName()) != 0  ||
            strcmp(driverName, Printer->GetSetup().GetDriverName()) != 0 ||
           strcmp(outputName, Printer->GetSetup().GetOutputName()) != 0 ) {
          // New printer installed so get the new printer device now.
          //
          delete Printer;
          Printer = new TPrinter(this);
        }
      }
      else {
        // No printer installed(GetProfileString failed).
        //
        delete Printer;
        Printer = new TPrinter(this);
      }
    }
  }
}


int OwlMain(int , char* [])
{
```

```
  GraphingApp   app;
  return app.Run();
}
```

```
//************************************************************************//
//  Project: User Interface (MDI) for 2D/3D TINs
//
//  SUBSYSTEM:    graphing.apx Application
//
//  FILE:         graphdocument.h
//
//
//
//  OVERVIEW
//
//  ~~~~~~~~
//
//  Class definition for GraphDocument (TFileDocument).
//
//  Author: Alias Abdul-Rahman (c) 1999
//
//************************************************************************//

#if !defined(graphdocument_h)  // Sentry, use file only if it's not already
#define graphdocument_h        //    included.

#include <owl/filedoc.h>

#include "graphingapp.rh"            // Definition of all resources.
#include <list>
#include <vector>


      struct Point
        {
         float x;
         float y;
         float z;
         int operator ==(const Point& p)
             const
                {
                 return x == p.x &&
                        y == p.y &&
                        z == p.z;
                }

         int operator <(const Point& p)
             const
               {
                return x < p.x &&
                       y < p.y &&
                       z < p.z;
               }
        };

      struct Triangle
        {
         int node_1;
         int node_2;
         int node_3;
         int operator ==(const Triangle& t)
             const
               {
                return node_1 == t.node_1 &&
                       node_2 == t.node_2 &&
```

```
                    node_3 == t.node_3;
              }

         int operator <(const Triangle& t)
             const
               {
                return node_1 < t.node_1 &&
                       node_2 < t.node_2 &&
                       node_3 < t.node_3;
               }
        };


     struct Segment
        {
         float x;
         float y;
         int Hreq;
         int SegNr;
         int operator ==(const Segment& seg)
             const
               {
                return x     == seg.x    &&
                       y     == seg.y    &&
                       Hreq  == seg.Hreq &&
                       SegNr == seg.SegNr;
               }

         int operator <(const Segment& seg)
             const
               {
                return x     < seg.x    &&
                       y     < seg.y    &&
                       Hreq  < seg.Hreq &&
                       SegNr < seg.SegNr;
               }
        };


//{{TFileDocument = GraphDocument}}
class GraphDocument : public TFileDocument {
  public:
    GraphDocument(TDocument* parent = 0);
    virtual ~GraphDocument();

//{{GraphDocumentVIRTUAL_BEGIN}}
  public:
    void GetXYZ_TIN()
      {
       if (!IsOpen())
         Open(ofRead) ;
      }

    virtual bool Open(int mode, const char far* path=0);
//{{GraphDocumentVIRTUAL_END}}

// Use STL containers to make life easier for
//      points, triangles, contour segments, etc.
  std::vector<Point> points;
  std::list<Triangle> triangles;
  std::list<Segment> contours;
```

```
  float xmin, ymin, xmax, ymax, zmin, zmax;
  };    //{{GraphDocument}}


#endif  // graphdocument_h sentry.
```

```
//****************************************************************************//
//  Project: User Interface (MDI) for 2D/3D TINs
//
//  SUBSYSTEM:    graphing.apx Application
//
//  FILE:         graphdocument.cpp
//
//
//
//  OVERVIEW
//
//  ~~~~~~~~
//
//  Source file for implementation of GraphDocument (TFileDocument).
//
//  Author: Alias Abdul-Rahman (c) 1999
//
//****************************************************************************//

#include <owl/pch.h>
#include <string.h>

#include "graphdocument.h"


//{{GraphDocument Implementation}}


GraphDocument::GraphDocument(TDocument* parent)
:
    TFileDocument(parent)
{
  // INSERT>> Your constructor code here.
}


GraphDocument::~GraphDocument()
{
  // INSERT>> Your destructor code here.
  Close();
}


bool GraphDocument::Open(int mode, const char far* path)
{
  bool result;
  // Read in all the data once
   if (path)
    SetDocPath(path);
  if (GetDocPath()) {
    TInStream* ist = InStream(ofRead);
    if (!ist)
      return false;

    bool ftt = true;
    Point p;

    string charX, charY, charZ;
    // read the file header
    *ist >> charX >> charY >> charZ;
```

```
  while (*ist) {
    *ist >> p.x >> p.y >> p.z;
    if (*ist) {
      points.push_back(p);
      // While we are here - get the min & max
      if (ftt)  {
        xmin = xmax = p.x;
        ymin = ymax = p.y;
        zmin = zmax = p.z;
        ftt = false;
      } else {
        xmin = min(xmin,p.x);
        ymin = min(ymin,p.y);
        zmin = min(zmin,p.z);
        xmax = max(xmax,p.x);
        ymax = max(ymax,p.y);
        zmax = max(zmax,p.z);
      }
    }
  }

  char t_path[MAX_PATH];
  strcpy(t_path, GetDocPath());
  // Nasty fudge to pick up matching TIN file
  for (int i = strlen(t_path) - 1; i > 0; i--) {
    if (t_path[i] == '.') {
     t_path[i + 1] = 'T';
     t_path[i + 2] = 'I';
     t_path[i + 3] = 'N';
     break;
    }
  }

  // read the file header of the TIN file
  string charNode1, charNode2 , charNode3;
  ifstream in_t(t_path);

  // Read the TINS
  Triangle t;

  in_t >> charNode1 >> charNode2 >> charNode3;

  while (in_t) {
    in_t >> t.node_1 >> t.node_2 >> t.node_3;
    if (in_t)
      triangles.push_back(t);

  }


 // to read the contour segment file
 char c_path[MAX_PATH];
  strcpy(c_path, GetDocPath());
  // Nasty fudge to pick up matching CON file
  for (int i = strlen(c_path) - 1; i > 0; i--) {
    if (c_path[i] == '.') {
     c_path[i + 1] = 'C';
     c_path[i + 2] = 'O';
     c_path[i + 3] = 'N';
     break;
    }
```

```
    }

    // read the header of the CON file
    string charXcon, charYcon, charHreq, charSegNr;
    ifstream in_c(c_path);

    // Read the Segments
    Segment seg;
    in_c >> charXcon >> charYcon >> charHreq >> charSegNr;

    while (in_c) {
      in_c >> seg.x >> seg.y >> seg.Hreq >> seg.SegNr;
      if (in_c)
        contours.push_back(seg);

    }

  delete ist;

  }
  return result;
}
```

```
//****************************************************************************//
//  Project: MDI for 2D/3D TINs
//
//
//  SUBSYSTEM:    Graphing Application
//
//  FILE:         apxprint.h
//
//  OVERVIEW
//
//  ~~~~~~~~
//
//  Class definition for TApxPrintout (TPrintout).
//
//  Author: Alias Abdul-Rahman (c) 1999
//
//****************************************************************************//

#if !defined(apxprint_h)   // Sentry use file only if it's not already
included.
#define apxprint_h

#include <owl/printer.h>


class TApxPrintout : public TPrintout {
  public:
    TApxPrintout(TPrinter* printer, const char far* title, TWindow* window,
                                                          bool scale = true)
      : TPrintout(title)
      {
        Printer = printer;
        Window = window;
        Scale = scale;
        MapMode = MM_ANISOTROPIC;
      }

    void GetDialogInfo(int& minPage, int& maxPage, int& selFromPage,
                       int& selToPage);
    void BeginPrinting();
    void BeginPage(TRect& clientR);
    void PrintPage(int page, TRect& rect, unsigned flags);
    void EndPage();
    void SetBanding(bool b)        { Banding = b; }
    bool HasPage(int pageNumber);

  protected:
    TWindow*  Window;
    bool      Scale;
    TPrinter* Printer;
    int       MapMode;

    int       PrevMode;
    TSize     OldVExt, OldWExt;
    TRect     OrgR;
};

#endif  // apxprint_h
```

```
//****************************************************************************//
//  Project: MDI for 2D/3D TINs
//
//  SUBSYSTEM:    Graphing Application
//
//  FILE:         apxprint.cpp
//
//  OVERVIEW
//
//  ~~~~~~~~
//
//  Source file for implementation of Printing.
//
//  Author: Alias Abdul-Rahman (c) 1999
//
//****************************************************************************//

#include <owl/pch.h>
#include "apxprint.h"


// Do not enable page range in the print dialog since only one page is
// available to be printed
//
void TApxPrintout::GetDialogInfo(int& minPage, int&, int& selFromPage, int&)
{
  minPage = 1;
  selFromPage = 0;
}


void TApxPrintout::BeginPrinting()
{
  TRect clientR;

  BeginPage(clientR);

  HFONT   hFont = (HFONT)Window->GetWindowFont();
  TFont   font("Arial", -12);
  if (!hFont)
    DC->SelectObject(font);
  else
    DC->SelectObject(TFont(hFont));

  TEXTMETRIC  tm;
  int fHeight = DC->GetTextMetrics(tm) ? tm.tmHeight + tm.tmExternalLeading
: 10;

  DC->RestoreFont();

  // How many lines of this font can we fit on a page.
  //
  int linesPerPage = MulDiv(clientR.Height(), 1, fHeight);

  TPrintDialog::TData& printerData = Printer->GetSetup();

   // GetMinMaxInfo event is overrided to return the number of lines when
printing.
  //
  MINMAXINFO minmaxinfo;
  Window->SendMessage(WM_GETMINMAXINFO, 0, (long)&minmaxinfo);
```

```
  int maxPg = (minmaxinfo.ptMaxSize.y / linesPerPage) + 1.0;

  // Compute the number of pages to print.
  //
  printerData.MinPage = 1;
  printerData.MaxPage = maxPg;

  EndPage();

  TPrintout::BeginPrinting();
}


void TApxPrintout::BeginPage(TRect& clientR)
{
  TScreenDC screenDC;
  TSize screenRes(screenDC.GetDeviceCaps(LOGPIXELSX),
          screenDC.GetDeviceCaps(LOGPIXELSY));
  TSize printRes(DC->GetDeviceCaps(LOGPIXELSX),
           DC->GetDeviceCaps(LOGPIXELSY));

  // Temporarily change the window size (so any WM_PAINT queries on the total
  // window size (GetClientRect) is
  // the window size for the WM_PAINT of the window and the printer page size
  // when Paint is called from
  // PrintPage. Notice, we don't use AdjustWindowRect because its harder and
  // not accurate.  Instead we
  // compute the difference (in pixels) between the client window and the
  // frame window.  This difference
  // is then added to the clientRect to compute the new frame window size
  // for SetWindowPos.
  //
  clientR = Window->GetClientRect();
  Window->MapWindowPoints(HWND_DESKTOP, (TPoint*)&clientR, 2);

  // Compute extra X and Y pixels to bring a client window dimensions to
  // equal the frame window.
  //
  OrgR = Window->GetWindowRect();
  int adjX = OrgR.Width() - clientR.Width();
  int adjY = OrgR.Height() - clientR.Height();

  // Conditionally scale the DC to the window so the printout will
  // resemble the window.
  //
  if (Scale) {
    clientR = Window->GetClientRect();
    PrevMode = DC->SetMapMode(MapMode);
    DC->SetViewportExt(PageSize, &OldVExt);

    // Scale window to logical page size (assumes left & top are 0)
    //
    clientR.right = MulDiv(PageSize.cx, screenRes.cx, printRes.cx);
    clientR.bottom = MulDiv(PageSize.cy, screenRes.cy, printRes.cy);

    DC->SetWindowExt(clientR.Size(), &OldWExt);
  }

  // Compute the new size of the window based on the printer DC dimensions.
  // Resize the window, notice position, order, and redraw are not done
  // the window size changes but the user
```

```
  // doesn't see any visible change to the window.
  //
  Window->SetRedraw(false);
   Window->SetWindowPos(0, 0, 0, clientR.Width() + adjX, clientR.Height() +
adjY,
                                SWP_NOMOVE | SWP_NOREDRAW | SWP_NOZORDER|
SWP_NOACTIVATE);
}


void TApxPrintout::PrintPage(int page, TRect& bandRect, unsigned)
{
  TRect clientR;

  BeginPage(clientR);

  if (Scale)
    DC->DPtoLP(bandRect, 2);

  // Change the printer range to this current page.
  //
  TPrintDialog::TData& printerData = Printer->GetSetup();
  int fromPg = printerData.FromPage;
  int toPg = printerData.ToPage;

  printerData.FromPage = page;
  printerData.ToPage = page;

  // Call the window to paint itself to the printer DC.
  //
  Window->Paint(*DC, false, bandRect);

  printerData.FromPage = fromPg;
  printerData.ToPage = toPg;

  if (Scale)
    DC->LPtoDP(bandRect, 2);

  EndPage();
}


void TApxPrintout::EndPage()
{
  // Resize to original window size, no one's the wiser.
  //
  Window->SetWindowPos(0, 0, 0, OrgR.Width(), OrgR.Height(),
               SWP_NOMOVE | SWP_NOREDRAW | SWP_NOZORDER| SWP_NOACTIVATE);
  Window->SetRedraw(true);

  // Restore changes made to the DC
  //
  if (Scale) {
    DC->SetWindowExt(OldWExt);
    DC->SetViewportExt(OldVExt);
    DC->SetMapMode(PrevMode);
  }
}


bool TApxPrintout::HasPage(int pageNumber)
```

```
{
  TPrintDialog::TData& printerData = Printer->GetSetup();

      return pageNumber >= printerData.MinPage && pageNumber <=
printerData.MaxPage;
}
```

```
//{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{//
//  Project: MDI for 2D/3D TINs
//
//  SUBSYSTEM:    Graphing Application
//
//  FILE:         graphingwindowview.h
//
//  OVERVIEW
//
//  ~~~~~~~~
//
//  Class definition for TGraphingWindowView (TWindowView).
//
//  Author: Alias Abdul-Rahman (c) 1999
//
//}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}//

#if !defined(graphingwindowview_h)   // Sentry, use file only if it's not
#define graphingwindowview_h         // ... already included.

#include <owl/docview.h>

#include "graphingapp.rh"            // Definition of all resources.


//{{TWindowView = TGraphingWindowView}}
class TGraphingWindowView : public TWindowView {
  public:
    TGraphingWindowView(TDocument& doc, TWindow* parent = 0);
    virtual ~TGraphingWindowView();

//{{TGraphingWindowViewVIRTUAL_BEGIN}}
  public:
    virtual void Paint(TDC& dc, bool erase, TRect& rect);
//{{TGraphingWindowViewVIRTUAL_END}}
//{{TGraphingWindowViewRSP_TBL_BEGIN}}
  protected:
    void EvGetMinMaxInfo(MINMAXINFO far& minmaxinfo);
    void EvSize(uint sizeType, TSize& size);
//{{TGraphingWindowViewRSP_TBL_END}}
DECLARE_RESPONSE_TABLE(TGraphingWindowView);
};    //{{TGraphingWindowView}}


#endif  // graphingwindowview_h sentry.
```

```
//*************************************************************************//
//  Project: User Interface (MDI) for 2D/3D TIN
//
//  SUBSYSTEM:    Graphing Application                                      //
//  FILE:         graphingwindowview.cpp                                   //
//  OVERVIEW                                                               //
//  ~~~~~~~~                                                               //
//  Source file for implementation of TGraphingWindowView (TWindowView).   //
//  Author: Alias Abdul-Rahman (c) 1999                                    //
//*************************************************************************//

#include <owl/pch.h>

#include "graphingapp.h"
#include "graphingwindowview.h"
#include "graphdocument.h"

#include <stdio.h>


//{{TGraphingWindowView Implementation}}


//
// Build a response table for all messages/commands handled
// by TGraphingWindowView derived from TWindowView.
//
DEFINE_RESPONSE_TABLE1(TGraphingWindowView, TWindowView)
//{{TGraphingWindowViewRSP_TBL_BEGIN}}
  EV_WM_GETMINMAXINFO,
  EV_WM_SIZE,
//{{TGraphingWindowViewRSP_TBL_END}}
END_RESPONSE_TABLE;


//----------------------------------------------------------
// TGraphingWindowView
// ~~~~~~~~~~~
// Construction/Destruction handling.
//
TGraphingWindowView::TGraphingWindowView(TDocument& doc, TWindow* parent)
  :
  TWindowView(doc, parent)
 {
  // INSERT>> Your constructor code here.
 }


TGraphingWindowView::~TGraphingWindowView()
 {
  // INSERT>> Your destructor code here.
 }


//
// Paint routine for Window, Printer, and PrintPreview for a TWindowView
client.
//
void TGraphingWindowView::Paint(TDC& dc, bool, TRect& rect)
 {
  HDC hdc;
```

```
  // to set the back ground color,
  // comment out the next line for default color
  //static const TColor color (RGB(192, 192, 192)); // RGB - light gray
  //TWindow::SetBkgndColor(color);
  GraphingApp* theApp = TYPESAFE_DOWNCAST(GetApplication(), GraphingApp);
  if (theApp)
    {
    // Only paint if we're printing and we have something to paint,
    // otherwise do nothing.
    //
    if (theApp->Printing && theApp->Printer && !rect.IsEmpty()) {
      // Use pageSize to get the size of the window to render into.
      // For a Window it's the client area,
      // for a printer it's the printer DC dimensions and for
      // print preview it's the layout window.
      //
      TSize   pageSize(rect.right - rect.left, rect.bottom - rect.top);

      TPrintDialog::TData& printerData = theApp->Printer->GetSetup();

      // Compute the number of pages to print.
      //
      printerData.MinPage = 1;
      printerData.MaxPage = 1;

      // INSERT>> Special printing code goes here.

    }
    else
     {
      // INSERT>> Normal painting code goes here.
     }
      GraphDocument& g_doc = dynamic_cast<GraphDocument&>(GetDocument());
      // Make sure the doc has read the points
      g_doc.GetXYZ_TIN();

      /*
      // Set the mapping mode so that all points fit on the scale
      dc.SetMapMode(MM_ANISOTROPIC);
            dc.SetWindowExt(TSize((g_doc.xmax  -  g_doc.xmin),(g_doc.ymax  -
g_doc.ymin)));
      dc.SetWindowOrg(TPoint(g_doc.xmin,g_doc.ymin));
      dc.SetViewportExt(GetClientRect().Size());
      dc.SetViewportOrg(TPoint(0, 0));
      */

      // set the device origin
      SetViewportOrgEx(dc, rect.left, rect.top, &POINT());

      float xlength = g_doc.xmax - g_doc.xmin;
      float ylength = g_doc.ymax - g_doc.ymin;

      float scaleX = (rect.right - rect.left) / xlength;
      float scaleY = (rect.bottom - rect.top) / ylength;


    // to handle points only
    TPen penpoints = TPen(RGB(0, 0, 255), 2, PS_SOLID);
    dc.SelectObject(penpoints);

    char s[16];
```

```
    // Use STL container to go through container in read order
    for (std::vector<Point>::iterator i = g_doc.points.begin();
                                       i != g_doc.points.end() ; i++)
     {
      int x = ((*i).x - g_doc.xmin) * scaleX;
      int y = (rect.bottom - rect.top) - ((*i).y - g_doc.ymin) * scaleY;
      int z = (*i).z;

      dc.MoveTo(x, y);
      dc.LineTo(x, y);
     }

// to handle triangles
#define TRIANGLES
// Comment out the above line to see the contours

#if defined TRIANGLES
     TPen pentri = TPen(RGB(0, 0, 255), 1, PS_SOLID);
     dc.SelectObject(pentri);

     for (std::list<Triangle>::iterator i = g_doc.triangles.begin();
                                        i != g_doc.triangles.end(); i++)
      {
       int xNd1 = (g_doc.points[(*i).node_1-1].x - g_doc.xmin) * scaleX;
       int yNd1 = (rect.bottom - rect.top) - (g_doc.points[(*i).node_1-1].y
                                          - g_doc.ymin) * scaleY;
       int xNd2 = (g_doc.points[(*i).node_2-1].x - g_doc.xmin) * scaleX;
       int yNd2 = (rect.bottom - rect.top) - (g_doc.points[(*i).node_2-1].y
                                          - g_doc.ymin) * scaleY;
       int xNd3 = (g_doc.points[(*i).node_3-1].x - g_doc.xmin) * scaleX;
       int yNd3 = (rect.bottom - rect.top) - (g_doc.points[(*i).node_3-1].y
                                          - g_doc.ymin) * scaleY;

       dc.MoveTo(xNd1, yNd1);
       dc.LineTo(xNd2, yNd2);
       dc.LineTo(xNd3, yNd3);
       dc.LineTo(xNd1, yNd1);
      }

#endif


// to handle Contour segments
#define CONTOURS
// Comment out the above line to see the contours

#if defined CONTOURS
     TPen pencont = TPen(RGB(255, 0, 0), 1, PS_SOLID);
     dc.SelectObject(pencont);

     int x = ((*g_doc.contours.begin()).x - g_doc.xmin) * scaleX;
     int y = (rect.bottom - rect.top) - ((*g_doc.contours.begin()).y
                                            - g_doc.ymin) * scaleY;
     int Hreq = (*g_doc.contours.begin()).Hreq;
     int SegNr = (*g_doc.contours.begin()).SegNr;
     int prev_Hreq = Hreq;
     int prev_SegNr = SegNr;
     int startx = x;
     int starty = y;
     dc.MoveTo(x, y);
```

```
     for (std::list<Segment>::iterator i = g_doc.contours.begin();
                                        i != g_doc.contours.end(); i++)
      {
       int nextx = ((*i).x - g_doc.xmin) * scaleX;
       int nexty = (rect.bottom - rect.top) - ((*i).y - g_doc.ymin) * scaleY;

       int nextHreq = (*i).Hreq;
       int nextSegNr = (*i).SegNr;

       if (SegNr != prev_SegNr)
         {
          dc.MoveTo(x, y);
          startx = x;
          starty = y;
         }
       else
         {
          if ((x == startx) && (y == starty))
             {
              if (SegNr == nextSegNr)
                 dc.MoveTo(x, y);
              else
                 dc.LineTo(x, y);
             }
          else
          dc.LineTo(x, y);
         }

       prev_SegNr = SegNr;
       SegNr = nextSegNr;

       x = nextx;
       y = nexty;
       Hreq = nextHreq;
      }
     dc.MoveTo(x, y);

#endif

     dc.RestorePen();
   }
 }


void TGraphingWindowView::EvGetMinMaxInfo(MINMAXINFO far& minmaxinfo)
 {
  GraphingApp* theApp = TYPESAFE_DOWNCAST(GetApplication(), GraphingApp);
  if (theApp)
    {
     if (theApp->Printing)
       {
        minmaxinfo.ptMaxSize = TPoint(32000, 32000);
        minmaxinfo.ptMaxTrackSize = TPoint(32000, 32000);
        return;
       }
    }
  TWindowView::EvGetMinMaxInfo(minmaxinfo);
 }

void TGraphingWindowView::EvSize(uint sizeType, TSize& size)
 {
```

```
    TWindowView::EvSize(sizeType, size);

    // INSERT>> Your code here.
    Invalidate();
}
```

```
//******************************************************************************//
//  Project: User Interface (MDI) for 2D/3D TINs
//
//  SUBSYSTEM:    Graphing Application
//
//  FILE:         graphingmdiclient.h
//
//  OVERVIEW
//
//  ~~~~~~~~
//
//  Class definition for TGraphingMDIClient (TMDIClient).
//
//  Author: Alias Abdul-Rahman (c) 1999
//
//******************************************************************************//

#if !defined(graphingmdiclient_h)    // Sentry, use file only if it's not
#define graphingmdiclient_h          // already included

#include "graphingapp.rh"            // Definition of all resources.


//{{TMDIClient = TGraphingMDIClient}}
class TGraphingMDIClient : public TMDIClient {
  public:
    int      ChildCount;                 // Number of child window created.

    TGraphingMDIClient(TModule* module = 0);
    virtual ~TGraphingMDIClient();

    void OpenFile(const char* fileName = 0);

//{{TGraphingMDIClientVIRTUAL_BEGIN}}
  protected:
    virtual void SetupWindow();
//{{TGraphingMDIClientVIRTUAL_END}}

//{{TGraphingMDIClientRSP_TBL_BEGIN}}
  protected:
    void CmFilePrint();
    void CmFilePrintSetup();
    void CmFilePrintPreview();
    void CmPrintEnable(TCommandEnabler& tce);
//{{TGraphingMDIClientRSP_TBL_END}}
DECLARE_RESPONSE_TABLE(TGraphingMDIClient);
};    //{{TGraphingMDIClient}}


#endif  // graphingmdiclient_h sentry.
```

```
//****************************************************************************//
//  Project: User Interface (MDI) for 2D/3D TINs
//
//  SUBSYSTEM:    Graphing Application
//
//  FILE:         graphingmdiclient.cpp
//
//  OVERVIEW
//
//  ~~~~~~~~
//
//  Source file for implementation of TGraphingMDIClient (TMDIClient).
//
//  Author: Alias Abdul-Rahman (c) 1999
//
//****************************************************************************//

#include <owl/pch.h>

#include <owl/docmanag.h>
#include <owl/listbox.h>
#include <stdio.h>

#include "graphingapp.h"
#include "graphingmdichild.h"
#include "graphingmdiclient.h"
#include "apxprint.h"
#include "apxprev.h"


//{{TGraphingMDIClient Implementation}}


//
// Build a response table for all messages/commands handled
// by TGraphingMDIClient derived from TMDIClient.
//
DEFINE_RESPONSE_TABLE1(TGraphingMDIClient, TMDIClient)
//{{TGraphingMDIClientRSP_TBL_BEGIN}}
  EV_COMMAND(CM_FILEPRINT, CmFilePrint),
  EV_COMMAND(CM_FILEPRINTERSETUP, CmFilePrintSetup),
  EV_COMMAND(CM_FILEPRINTPREVIEW, CmFilePrintPreview),
  EV_COMMAND_ENABLE(CM_FILEPRINT, CmPrintEnable),
  EV_COMMAND_ENABLE(CM_FILEPRINTERSETUP, CmPrintEnable),
  EV_COMMAND_ENABLE(CM_FILEPRINTPREVIEW, CmPrintEnable),
//{{TGraphingMDIClientRSP_TBL_END}}
END_RESPONSE_TABLE;


//----------------------------------------------------------
// TGraphingMDIClient
// ~~~~~~~~~~~~
// Construction/Destruction handling.
//
TGraphingMDIClient::TGraphingMDIClient(TModule* module)
:
  TMDIClient(module)
{
  ChildCount = 0;

  // INSERT>> Your constructor code here.
```

```
}


TGraphingMDIClient::~TGraphingMDIClient()
{
  Destroy();

  // INSERT>> Your destructor code here.

}


//----------------------------------------------------------
// TGraphingMDIClient
// ~~~~~~~~~~~~
// MDIClient site initialization.
//
void TGraphingMDIClient::SetupWindow()
{
  // Default SetUpWindow processing.
  //
  TMDIClient::SetupWindow();

}


//----------------------------------------------------------
// TGraphingMDIClient
// ~~~~~~~~~~
// Menu File Print command
//
void TGraphingMDIClient::CmFilePrint()
{
  // Create Printer object if not already created.
  //
  GraphingApp* theApp = TYPESAFE_DOWNCAST(GetApplication(), GraphingApp);
  if (theApp) {
    if (!theApp->Printer)
      theApp->Printer = new TPrinter(theApp);

    TAPointer<char> docName = new char[_MAX_PATH];

    TDocument* currentDoc = theApp->GetDocManager()->GetCurrentDoc();
    if (currentDoc->GetTitle())
      strcpy(docName, currentDoc->GetTitle());
    else
      strcpy(docName, "Document");

    // Create Printout window and set characteristics.
    //
    TApxPrintout printout(theApp->Printer, docName,
                          GetActiveMDIChild()->GetClientWindow(), true);

    theApp->Printing++;

    // Bring up the Print dialog and print the document.
    //
    theApp->Printer->Print(GetWindowPtr(GetActiveWindow()), printout, true);

    theApp->Printing--;
  }
```

```
}


//----------------------------------------------------------
// TGraphingMDIClient
// ~~~~~~~~~~
// Menu File Print Setup command
//
void TGraphingMDIClient::CmFilePrintSetup()
{
  GraphingApp* theApp = TYPESAFE_DOWNCAST(GetApplication(), GraphingApp);
  if (theApp) {
    if (!theApp->Printer)
      theApp->Printer = new TPrinter(theApp);

    // Bring up the Print Setup dialog.
    //
    theApp->Printer->Setup(this);
  }
}


//----------------------------------------------------------
// TGraphingMDIClient
// ~~~~~~~~~~
// Menu File Print Preview command
//
void TGraphingMDIClient::CmFilePrintPreview()
{
  GraphingApp* theApp = TYPESAFE_DOWNCAST(GetApplication(), GraphingApp);
  if (theApp) {
    if (!theApp->Printer)
      theApp->Printer = new TPrinter(GetApplication());

    theApp->Printing++;

    TApxPreviewWin* prevW = new TApxPreviewWin(Parent, theApp->Printer,
                            GetActiveMDIChild()->GetClientWindow(),
                            "Print Preview", new TLayoutWindow(0));
    prevW->Create();

    // Here we resize the preview window to take the size of the MainWindow,
    // then hide the MainWindow.
    //
    TFrameWindow * mainWindow =  GetApplication()->GetMainWindow();
    TRect r = mainWindow->GetWindowRect();
    prevW->MoveWindow(r);
    prevW->ShowWindow(SW_SHOWNORMAL);
    mainWindow->ShowWindow(SW_HIDE);

    GetApplication()->BeginModal(GetApplication()->GetMainWindow());

    // After the user closes the preview Window, we take its size and use it
    // to size the MainWindow, then show the MainWindow again.
    //
    r = prevW->GetWindowRect();
    mainWindow->MoveWindow(r);
    mainWindow->ShowWindow(SW_SHOWNORMAL);

    // We must destroy the preview window explicitly.  Otherwise, the window
will
```

```
    // not be destroyed until it's parent the MainWindow is destroyed.
    //
    prevW->Destroy();
    delete prevW;

    theApp->Printing--;
  }
}


//----------------------------------------------------------
// TGraphingMDIClient
// ~~~~~~~~~~
// Menu enabler used by Print, Print Setup and Print Preview.
//
void TGraphingMDIClient::CmPrintEnable(TCommandEnabler& tce)
{
  if (GetActiveMDIChild()) {
    GraphingApp* theApp = TYPESAFE_DOWNCAST(GetApplication(), GraphingApp);
    if (theApp) {
      // If we have a Printer already created just test if all is okay.
      // Otherwise create a Printer object and make sure the printer really
      // exists and then delete the Printer object.
      //
      if (!theApp->Printer) {
        theApp->Printer = new TPrinter(theApp);
        tce.Enable(!theApp->Printer->GetSetup().Error);
      }
      else
        tce.Enable(!theApp->Printer->GetSetup().Error);
    }
  }
  else
    tce.Enable(false);
}
```

```
//****************************************************************************//
//  Project: User Interface (MDI) for 2D/3D TINs
//
//  SUBSYSTEM:    Graphing Application
//
//  FILE:         graphingmdichild.h
//
//
//  OVERVIEW
//
//  ~~~~~~~~
//
//  Class definition for TGraphingMDIChild (TMDIChild).
//
//  Author : Alias Abdul-Rahman  (c) 1999
//
//****************************************************************************//

#if !defined(graphingmdichild_h)     // Sentry, use file only if it's not
#define graphingmdichild_h           // not alraedy included

#include "graphingapp.rh"            // Definition of all resources.


//{{TMDIChild = TGraphingMDIChild}}
class TGraphingMDIChild : public TMDIChild {
  public:
    TGraphingMDIChild(TMDIClient& parent, const char far* title,
                      TWindow* clientWnd, bool shrinkToClient = false,
                      TModule* module = 0);
    virtual ~TGraphingMDIChild();
};    //{{TGraphingMDIChild}}


#endif  // graphingmdichild_h sentry.
```

```
//****************************************************************************//
//  Project: MDI for 2D/3D TINs
//
//  SUBSYSTEM:    Graphing Application
//
//  FILE:         graphingmdichild.cpp
//
//  OVERVIEW
//
//  ~~~~~~~~
//
//  Source file for implementation of TGraphingMDIChild (TMDIChild).
//
//  Author : Alias Abdul-Rahman (c) 1999
//
//****************************************************************************//

#include <owl/pch.h>

#include "graphingapp.h"
#include "graphingmdichild.h"


//{{TGraphingMDIChild Implementation}}


//----------------------------------------------------------
// TGraphingMDIChild
// ~~~~~~~~~~
// Construction/Destruction handling.
//
TGraphingMDIChild::TGraphingMDIChild(TMDIClient& parent,
                                     const char far* title, TWindow*
clientWnd,
                                     bool shrinkToClient, TModule* module)
:
  TMDIChild(parent, title, clientWnd, shrinkToClient, module)
{
  // INSERT>> Your constructor code here.

}


TGraphingMDIChild::~TGraphingMDIChild()
{
  Destroy();

  // INSERT>> Your destructor code here.

}
```

```
//************************************************************************//
//  Project: User Interface (MDI) for 2D/3D TINs
//
//  SUBSYSTEM:    Graphing Application                                     //
//  FILE:         apxprev.h
//
//  OVERVIEW                                                              //
//  ~~~~~~~~                                                              //
//  Class definition for TApxPreviewWin (Print Preview).                  //
//  Author: Alias Abdul-Rahman (c) 1999                                   //
//************************************************************************//

#if !defined(apxprev_h)    // Sentry, use file only if it's not already
included.
#define apxprev_h

#include <owl/controlb.h>
#include <owl/preview.h>

#include "apxprint.h"
#include "graphingapp.rh"


//{{TDecoratedFrame = TApxPreviewWin}}
class TApxPreviewWin : public TDecoratedFrame {
  public:
        TApxPreviewWin(TWindow*  parentWindow,  TPrinter*  printer,  TWindow*
currWindow,
                  const char far* title, TLayoutWindow* client);
   ~TApxPreviewWin();

    int             PageNumber, FromPage, ToPage;

    TWindow*        CurrWindow;
    TControlBar*    PreviewSpeedBar;
    TPreviewPage*   Page1;
    TPreviewPage*   Page2;
    TPrinter*       Printer;

    TPrintDC*       PrnDC;
    TSize*          PrintExtent;
    TApxPrintout*   Printout;

  private:
    TLayoutWindow*  Client;

    void SpeedBarState();
    void PPR_PreviousEnable(TCommandEnabler& tce);
    void PPR_NextEnable(TCommandEnabler& tce);
    void PPR_Previous();
    void PPR_Next();
    void PPR_OneUp();
    void PPR_TwoUpEnable(TCommandEnabler& tce);
    void PPR_TwoUp();
    void PPR_Done();
    void CmPrintEnable(TCommandEnabler& tce);
    void CmPrint();

//{{TApxPreviewWinVIRTUAL_BEGIN}}
  protected:
    virtual void SetupWindow();
```

```
//{{TApxPreviewWinVIRTUAL_END}}

//{{TApxPreviewWinRSP_TBL_BEGIN}}
  protected:
    void EvClose();
//{{TApxPreviewWinRSP_TBL_END}}
DECLARE_RESPONSE_TABLE(TApxPreviewWin);
};    //{{TApxPreviewWin}}

#endif  // apxprev_h
```

```
//****************************************************************************//
//  Project: User Interface (MDI) for 2D/3D TINs
//
//  SUBSYSTEM:    Graphing Application
//
//  FILE:         apxprev.cpp
//
//  OVERVIEW
//
//  ~~~~~~~~
//
//  Source file for implementation of Print Preview.
//
//  Author : Alias Abdul-Rahman (c) 1999
//
//****************************************************************************//

#include <owl/pch.h>

#include <owl/buttonga.h>
#include <owl/textgadg.h>
#include <stdio.h>

#include "apxprev.h"
#include "graphingapp.rh"


//{{TApxPreviewWin Implementation}}

DEFINE_RESPONSE_TABLE1(TApxPreviewWin, TDecoratedFrame)
  EV_COMMAND_ENABLE(APX_PPR_PREVIOUS, PPR_PreviousEnable),
  EV_COMMAND_ENABLE(APX_PPR_NEXT, PPR_NextEnable),
  EV_COMMAND(APX_PPR_PREVIOUS, PPR_Previous),
  EV_COMMAND(APX_PPR_NEXT, PPR_Next),
  EV_COMMAND(APX_PPR_ONEUP, PPR_OneUp),
  EV_COMMAND_ENABLE(APX_PPR_TWOUP, PPR_TwoUpEnable),
  EV_COMMAND(APX_PPR_TWOUP, PPR_TwoUp),
  EV_COMMAND(APX_PPR_DONE, PPR_Done),
  EV_COMMAND(CM_FILEPRINT, CmPrint),
  EV_COMMAND_ENABLE(CM_FILEPRINT, CmPrintEnable),
//{{TApxPreviewWinRSP_TBL_BEGIN}}
  EV_WM_CLOSE,
//{{TApxPreviewWinRSP_TBL_END}}
END_RESPONSE_TABLE;


TApxPreviewWin::TApxPreviewWin(TWindow* parentWindow, TPrinter* printer,
                               TWindow* currWindow, const char far* title,
                               TLayoutWindow* client)
:
  TDecoratedFrame(parentWindow, title, client)
{
  CurrWindow = currWindow;
  Printer = printer;
  Client = client;
  Page1 = 0;
  Page2 = 0;
  FromPage = 1;
  ToPage = 1;

  TPrintDialog::TData& data = Printer->GetSetup();
```

```
  PrnDC = new TPrintDC(data.GetDriverName(),
                       data.GetDeviceName(),
                       data.GetOutputName(),
                       data.GetDevMode());

  PrintExtent = new TSize(PrnDC->GetDeviceCaps(HORZRES),
                          PrnDC->GetDeviceCaps(VERTRES));
  Printout = new TApxPrintout(Printer, "Print Preview", currWindow, true);

  SetBkgndColor(::GetSysColor(COLOR_APPWORKSPACE));

  // Create default toolbar New and associate toolbar buttons with commands.
  //
  PreviewSpeedBar = new TControlBar(this);
        PreviewSpeedBar->Insert(*new    TButtonGadget(APX_PPR_PREVIOUS,
APX_PPR_PREVIOUS,
                           TButtonGadget::Command, true));
  PreviewSpeedBar->Insert(*new TButtonGadget(APX_PPR_NEXT, APX_PPR_NEXT,
                           TButtonGadget::Command, true));
  PreviewSpeedBar->Insert(*new TSeparatorGadget(6));
  PreviewSpeedBar->Insert(*new TButtonGadget(APX_PPR_ONEUP, APX_PPR_ONEUP,
                                          TButtonGadget::Exclusive,  true,
TButtonGadget::Down));
  PreviewSpeedBar->Insert(*new TButtonGadget(APX_PPR_TWOUP, APX_PPR_TWOUP,
                           TButtonGadget::Exclusive, true));
  PreviewSpeedBar->Insert(*new TSeparatorGadget(12));
        PreviewSpeedBar->Insert(*new    TTextGadget(APX_PPR_CURRPAGE,
TGadget::Recessed,
                           TTextGadget::Left, 10, "Page 1"));
  PreviewSpeedBar->Insert(*new TSeparatorGadget(20));
  PreviewSpeedBar->Insert(*new TButtonGadget(CM_FILEPRINT, CM_FILEPRINT,
                           TButtonGadget::Command, true));
  PreviewSpeedBar->Insert(*new TSeparatorGadget(20));
  PreviewSpeedBar->Insert(*new TButtonGadget(APX_PPR_DONE, APX_PPR_DONE,
                           TButtonGadget::Command, true));
  Insert(*PreviewSpeedBar, TDecoratedFrame::Top);

  Attr.Style &= ~WS_VISIBLE;
  SetAcceleratorTable(IDM_PRINTPREVIEW);
}


TApxPreviewWin::~TApxPreviewWin()
{
  delete Page1;
  delete Page2;

  delete PrnDC;
  delete PrintExtent;
  delete Printout;
}


void TApxPreviewWin::SetupWindow()
{
  TDecoratedFrame::SetupWindow();

  TAPointer<char> captionText = new char[256];

  // Set the caption of the preview window based on that of the Main Window.
```

```
  //
  GetApplication()->GetMainWindow()->GetWindowText(captionText, 256);
  strcat(captionText, " (Preview)");
  SetCaption(captionText);

  // Set the icons of the preview window.
  //
  SetIcon(GetApplication(), IDI_MDIAPPLICATION);
  SetIconSm(GetApplication(), IDI_MDIAPPLICATION);

  TPrintDialog::TData& data = Printer->GetSetup();
  Page1 = new TPreviewPage(Client, *Printout, *PrnDC, *PrintExtent, 1);
  Page1->SetPageNumber(1);
  data.MaxPage = 1;

  Page2 = 0;

  TLayoutMetrics metrics1;

  metrics1.X.Set(lmLeft, lmRightOf, lmParent, lmLeft, 15);
  metrics1.Y.Set(lmTop, lmBelow, lmParent, lmTop, 15);

  // Determine major axis of preview page, have that follow parent size.
  // Make minor axis a percentage (aspect ratio) of the page's major axis
  //
  TRect r = Client->GetClientRect();
  long ratio;

  if (PrintExtent->cx > PrintExtent->cy)
    ratio = ((long)PrintExtent->cy * 100) / PrintExtent->cx;
  else
    ratio = ((long)PrintExtent->cx * 100) / PrintExtent->cy;

  bool xMajor = ((r.Width() * ratio) / 100) > r.Height();
  if (xMajor){
    metrics1.Height.Set(lmBottom, lmAbove, lmParent, lmBottom, 15);
    metrics1.Width.PercentOf(Page1,
                                        (int)((long)PrintExtent->cx * 95 /
PrintExtent->cy),
                             lmHeight);
  }
  else {
    metrics1.Height.PercentOf(Page1,
                                        (int)((long)PrintExtent->cy * 95 /
PrintExtent->cx),
                              lmWidth);
    metrics1.Width.Set(lmRight, lmLeftOf, lmParent, lmRight, 15);
  }

  Page1->Create();

  Client->SetChildLayoutMetrics(*Page1, metrics1);
  Client->Layout();
}


void TApxPreviewWin::SpeedBarState()
{
  // Update the page count.
  //
  TTextGadget* cpGadget = TYPESAFE_DOWNCAST
```

```
                          (PreviewSpeedBar->GadgetWithId(APX_PPR_CURRPAGE),
                           TTextGadget);
  if (cpGadget) {
    TAPointer<char> buffer = new char[32];

    if (Page2 && FromPage != ToPage)
      sprintf(buffer, "Page %d - %d", FromPage, ToPage);
    else
      sprintf(buffer, "Page %d", FromPage);
    cpGadget->SetText(buffer);
  }
}


void TApxPreviewWin::PPR_PreviousEnable(TCommandEnabler& tce)
{
  // Only have previous on if we're not at the first page.
  //
  tce.Enable(FromPage != 1);
}


void TApxPreviewWin::PPR_NextEnable(TCommandEnabler& tce)
{
  // Only have next on if we're not at the last page.
  //
  TPrintDialog::TData& printerData = Printer->GetSetup();
  tce.Enable(ToPage != printerData.MaxPage);
}


void TApxPreviewWin::PPR_Previous()
{
  TPrintDialog::TData& printerData = Printer->GetSetup();

  if (FromPage > printerData.MinPage) {
    FromPage--;
    ToPage--;

    Page1->SetPageNumber(FromPage);
    if (Page2)
      Page2->SetPageNumber(ToPage);
  }

  SpeedBarState();
}


void TApxPreviewWin::PPR_Next()
{
  TPrintDialog::TData& printerData = Printer->GetSetup();

  if (ToPage < printerData.MaxPage) {
    FromPage++;
    ToPage++;

    Page1->SetPageNumber(FromPage);
    if (Page2)
      Page2->SetPageNumber(ToPage);
  }
```

```
  SpeedBarState();
}


void TApxPreviewWin::PPR_OneUp()
{
  if (Page2) {
    Client->RemoveChildLayoutMetrics(*Page2);

    delete Page2;
    Page2 = 0;

    Client->Layout();

    ToPage = FromPage;

    SpeedBarState();
  }
}


void TApxPreviewWin::PPR_TwoUpEnable(TCommandEnabler& tce)
{
  // Two up is only available for portrait mode.
  //
  tce.Enable(PrintExtent->cx <= PrintExtent->cy);
}


void TApxPreviewWin::PPR_TwoUp()
{
  if (!Page2) {
    Page2 = new TPreviewPage(Client, *Printout, *PrnDC, *PrintExtent,
                             PageNumber + 1);
    Page2->Create();

    TLayoutMetrics metrics2;

    metrics2.X.Set(lmLeft, lmRightOf, Page1, lmRight, 30);
    metrics2.Y.SameAs(Page1, lmTop);

    // Assume portrait
    //
    metrics2.Width.SameAs(Page1, lmWidth);
    metrics2.Height.SameAs(Page1, lmBottom);

    Client->SetChildLayoutMetrics(*Page2, metrics2);
    Client->Layout();

    TPrintDialog::TData& printerData = Printer->GetSetup();

    // Page 2 is the next page.  If the next page is outside of our range then
    // set the first page back one and the 2nd page is the current page. If
the
    // document is only 1 page long then the 2nd page is empty.
    //
    if (FromPage == printerData.MaxPage) {
      if (FromPage > 1) {
        FromPage--;
        ToPage = FromPage + 1;
        Page1->SetPageNumber(FromPage);
```

```
        Page2->SetPageNumber(ToPage);
      }
      else
        Page2->SetPageNumber(0);
    }
    else {
      ToPage = FromPage + 1;
      Page2->SetPageNumber(ToPage);
    }

    SpeedBarState();
  }
}


void TApxPreviewWin::PPR_Done()
{
  // Don't call the base class EvClose; we do not want TApxPreviewWin to be
  // destructed.
  //
  GetApplication()->EndModal(IDCANCEL);
}


void TApxPreviewWin::EvClose()
{
  // Don't call the base class EvClose; we do not want TApxPreviewWin to be
  // destructed.
  //
  GetApplication()->EndModal(IDCANCEL);
}


void TApxPreviewWin::CmPrint()
{
  TWindow* client = GetApplication()->GetMainWindow()->GetClientWindow();

  if (client)
    client->SendMessage(WM_COMMAND, CM_FILEPRINT, 0);
}


void TApxPreviewWin::CmPrintEnable(TCommandEnabler& tce)
{
  tce.Enable(true);
}
```

```
//****************************************************************//
//  Project: User Interface (MDI) for 2D/3D TINs
//
//  SUBSYSTEM:    Graphing Application
//
//  FILE:         graphingaboutdlg.h
//
//
//  OVERVIEW
//
//  ~~~~~~~~
//
//  Class definition for GraphingAboutDlg (TDialog).
//
//  Author: Alias Abdul-Rahman (c) 1999
//
//****************************************************************//

#if !defined(graphingaboutdlg_h)   // Sentry, use file only if it's not
#define graphingaboutdlg_h         // already included.

#include <owl/static.h>

#include "graphingapp.rh"          // Definition of all resources.


//{{TDialog = GraphingAboutDlg}}
class GraphingAboutDlg : public TDialog {
  public:
    GraphingAboutDlg(TWindow* parent, TResId resId = IDD_ABOUT,
                     TModule* module = 0);
    virtual ~GraphingAboutDlg();

//{{GraphingAboutDlgVIRTUAL_BEGIN}}
  public:
    void SetupWindow();
//{{GraphingAboutDlgVIRTUAL_END}}
};    //{{GraphingAboutDlg}}


// Reading the VERSIONINFO resource.
//
class TProjectRCVersion {
  public:
    TProjectRCVersion(TModule* module);
    virtual ~TProjectRCVersion();

    bool GetProductName(LPSTR& prodName);
    bool GetProductVersion(LPSTR& prodVersion);
    bool GetCopyright(LPSTR& copyright);
    bool GetDebug(LPSTR& debug);

  protected:
    uint8 far*  TransBlock;
    void far*   FVData;

  private:
    // Don't allow this object to be copied.
    //
    TProjectRCVersion(const TProjectRCVersion&);
    TProjectRCVersion& operator = (const TProjectRCVersion&);
```

```
};

#endif  // graphingaboutdlg_h sentry.
```

```
//****************************************************************************//
//  Project: User Interface (MDI) for 2D/3D TINs
//
//  SUBSYSTEM:    Graphing Application
//
//  FILE:         graphingaboutdlg.cpp
//
//  OVERVIEW
//
//  ~~~~~~~~
//
//  Source file for implementation of GraphingAboutDlg (TDialog).
//
//  Author : Alias Abdul-Rahman (c) 1999
//
//****************************************************************************//

#include <owl/pch.h>
#include <stdio.h>
#if defined(BI_PLAT_WIN16)
# include <ver.h>
#endif

#include "graphingapp.h"
#include "graphingaboutdlg.h"


TProjectRCVersion::TProjectRCVersion(TModule* module)
{
  uint32  fvHandle;
  uint    vSize;
  char    appFName[255];
  TAPointer<char> subBlockName = new char[255];

  FVData = 0;

  module->GetModuleFileName(appFName, sizeof appFName);
  OemToAnsi(appFName, appFName);
  uint32 dwSize = ::GetFileVersionInfoSize(appFName, &fvHandle);
  if (dwSize) {
    FVData  = (void far *)new char[(uint)dwSize];
    if (::GetFileVersionInfo(appFName, fvHandle, dwSize, FVData)) {
      // Copy string to buffer so if the -dc compiler switch(Put constant
      // strings in code segments)
      // is on VerQueryValue will work under Win16.  This works around
      // a problem in Microsoft's ver.dll
      // which writes to the string pointed to by subBlockName.
      //
      strcpy(subBlockName, "\\VarFileInfo\\Translation");
      if (!::VerQueryValue(FVData, subBlockName,(void far* far*)&TransBlock,
          &vSize)) {
         delete[] FVData;
         FVData = 0;
       }
      else
        // Swap the words so sprintf will print the lang-charset in the
        // correct format.
        //
        *(uint32 *)TransBlock = MAKELONG(HIWORD(*(uint32 *)TransBlock),
                                         LOWORD(*(uint32 *)TransBlock));
    }
```

```
  }
}


TProjectRCVersion::~TProjectRCVersion()
{
  if (FVData)
    delete[] FVData;
}


bool TProjectRCVersion::GetProductName(LPSTR& prodName)
{
  uint    vSize;
  TAPointer<char> subBlockName = new char[255];

  if (FVData) {
          sprintf(subBlockName,   "\\StringFileInfo\\%08lx\\%s",   *(uint32
*)TransBlock,
           (LPSTR)"ProductName");
    return FVData ? ::VerQueryValue(FVData, subBlockName,
                                        (void far* far*)&prodName, &vSize) :
false;
  } else
    return false;
}


bool TProjectRCVersion::GetProductVersion(LPSTR& prodVersion)
{
  uint    vSize;
  TAPointer<char> subBlockName = new char[255];

  if (FVData) {
          sprintf(subBlockName,   "\\StringFileInfo\\%08lx\\%s",   *(uint32
*)TransBlock,
                           (LPSTR)"ProductVersion");
    return FVData ? ::VerQueryValue(FVData, subBlockName,
                              (void far* far*)&prodVersion, &vSize) : false;
  } else
    return false;
}


bool TProjectRCVersion::GetCopyright(LPSTR& copyright)
{
  uint    vSize;
  TAPointer<char> subBlockName = new char[255];

  if (FVData) {
          sprintf(subBlockName,   "\\StringFileInfo\\%08lx\\%s",   *(uint32
*)TransBlock,
                           (LPSTR)"LegalCopyright");
    return FVData ? ::VerQueryValue(FVData, subBlockName,
                                        (void far* far*)&copyright, &vSize) :
false;
  } else
    return false;
}
```

```
bool TProjectRCVersion::GetDebug(LPSTR& debug)
{
  uint    vSize;
  TAPointer<char> subBlockName = new char[255];

  if (FVData) {
           sprintf(subBlockName,   "\\StringFileInfo\\%08lx\\%s",   *(uint32
*)TransBlock,
                           (LPSTR)"SpecialBuild");
    return FVData ? ::VerQueryValue(FVData, subBlockName,
                                    (void far* far*)&debug, &vSize) : false;
  } else
    return false;
}


//{{GraphingAboutDlg Implementation}}


//----------------------------------------------------------
// GraphingAboutDlg
// ~~~~~~~~~~
// Construction/Destruction handling.
//
GraphingAboutDlg::GraphingAboutDlg(TWindow* parent, TResId resId, TModule*
module)
:
  TDialog(parent, resId, module)
{
  // INSERT>> Your constructor code here.
}


GraphingAboutDlg::~GraphingAboutDlg()
{
  Destroy();

  // INSERT>> Your destructor code here.
}


void GraphingAboutDlg::SetupWindow()
{
  LPSTR prodName = 0, prodVersion = 0, copyright = 0, debug = 0;

  // Get the static text for the value based on VERSIONINFO.
  //
  TStatic* versionCtrl = new TStatic(this, IDC_VERSION, 255);
  TStatic* copyrightCtrl = new TStatic(this, IDC_COPYRIGHT, 255);
  TStatic* debugCtrl = new TStatic(this, IDC_DEBUG, 255);

  TDialog::SetupWindow();

  // Process the VERSIONINFO.
  //
  TProjectRCVersion applVersion(GetModule());

  // Get the product name and product version strings.
  //
  if (applVersion.GetProductName(prodName) &&
                           applVersion.GetProductVersion(prodVersion)) {
```

```
    // IDC_VERSION is the product name and version number, the initial value
    // of IDC_VERSION is
    // the word Version(in whatever language) product name VERSION
    // product version.
    //
    char buffer[255];
    char versionName[128];

    buffer[0] = '\0';
    versionName[0] = '\0';

    versionCtrl->GetText(versionName, sizeof versionName);
    sprintf(buffer, "%s %s %s", prodName, versionName, prodVersion);

    versionCtrl->SetText(buffer);
  }

  // Get the legal copyright string.
  //
  if (applVersion.GetCopyright(copyright))
    copyrightCtrl->SetText(copyright);

  // Only get the SpecialBuild text if the VERSIONINFO resource is there.
  //
  if (applVersion.GetDebug(debug))
    debugCtrl->SetText(debug);
}
```

```
//************************************************************************//
//  Project: User Interface (MDI) for 2D/3D TINs                           //
//                                                                         //
//  SUBSYSTEM:  Graphing Application                                       //
//  FILE:     graphingapp.rc                                               //
//  AUTHOR:                                                                //
//                                                                         //
//  OVERVIEW                                                               //
//  ~~~~~~~~                                                               //
//  All resources defined here.                                            //
//                                                                         //
//  Author: Alias Abdul-Rahman (c) 1999                                    //
//************************************************************************//

#if !defined(WORKSHOP_INVOKED)
# include <windows.h>
#endif
#include "graphingapp.rh"


IDM_MDI MENU
BEGIN
  POPUP "&File"
  BEGIN
    MENUITEM "&New", CM_MDIFILENEW
    MENUITEM "&Open Input Files...", CM_MDIFILEOPEN
    MENUITEM "&Close", CM_FILECLOSE
    MENUITEM SEPARATOR
    MENUITEM "&Save", CM_FILESAVE, GRAYED
    MENUITEM "Save &As...", CM_FILESAVEAS, GRAYED
    MENUITEM SEPARATOR
    MENUITEM "Print Pre&view...", CM_FILEPRINTPREVIEW, GRAYED
    MENUITEM "&Print...", CM_FILEPRINT, GRAYED
    MENUITEM "P&rint Setup...", CM_FILEPRINTERSETUP, GRAYED
    MENUITEM SEPARATOR
    MENUITEM "E&xit\tAlt+F4", CM_EXIT
  END

  MENUITEM SEPARATOR

  //POPUP "&Edit"
  //BEGIN
    //MENUITEM "&Undo\tAlt+BkSp", CM_EDITUNDO, GRAYED
    //MENUITEM SEPARATOR
    //MENUITEM "Cu&t\tShift+Del", CM_EDITCUT, GRAYED
    //MENUITEM "&Copy\tCtrl+Ins", CM_EDITCOPY, GRAYED
    //MENUITEM "&Paste\tShift+Ins", CM_EDITPASTE, GRAYED
    //MENUITEM SEPARATOR
    //MENUITEM "Clear &All\tCtrl+Del", CM_EDITCLEAR, GRAYED
    //MENUITEM "&Delete\tDel", CM_EDITDELETE, GRAYED
  //END

  //POPUP "&Search"
  //BEGIN
    //MENUITEM "&Find...", CM_EDITFIND, GRAYED
    //MENUITEM "&Replace...", CM_EDITREPLACE, GRAYED
    //MENUITEM "&Next\aF3", CM_EDITFINDNEXT, GRAYED
  //END

  MENUITEM SEPARATOR
```

```
  MENUITEM SEPARATOR

  MENUITEM SEPARATOR

  POPUP "&Window"
  BEGIN
    MENUITEM "&Cascade", CM_CASCADECHILDREN
    MENUITEM "&Tile", CM_TILECHILDREN
    MENUITEM "Arrange &Icons", CM_ARRANGEICONS
    MENUITEM "C&lose All", CM_CLOSECHILDREN
  END

  MENUITEM SEPARATOR

  POPUP "&Help"
  BEGIN
    MENUITEM "&About...", CM_HELPABOUT
  END

END


// Accelerator table for short-cut to menu commands. (include/owl/editfile.rc)
//
//IDM_MDI ACCELERATORS
//BEGIN
  //VK_DELETE, CM_EDITCUT, VIRTKEY, SHIFT
  //VK_INSERT, CM_EDITCOPY, VIRTKEY, CONTROL
  //VK_INSERT, CM_EDITPASTE, VIRTKEY, SHIFT
  //VK_DELETE, CM_EDITCLEAR, VIRTKEY, CONTROL
  //VK_BACK,   CM_EDITUNDO, VIRTKEY, ALT
  //VK_F3,     CM_EDITFINDNEXT, VIRTKEY
//END


// Accelerator table for Print Preview window.
//
IDM_PRINTPREVIEW ACCELERATORS
BEGIN
  VK_ESCAPE, APX_PPR_DONE, VIRTKEY
  "c",       APX_PPR_DONE, ALT
END



// Menu merged in when TEditView is active, notice the extra MENUITEM
SEPARATORs which are
// for menu negotation.  These separators are used as group markers by OWL.
//
IDM_EDITVIEW MENU LOADONCALL MOVEABLE PURE DISCARDABLE
BEGIN
  MENUITEM SEPARATOR

  //POPUP "&Edit"
  //BEGIN
    //MENUITEM "&Undo\aCtrl+Z", CM_EDITUNDO
    //MENUITEM SEPARATOR
    //MENUITEM "&Cut\aCtrl+X", CM_EDITCUT
    //MENUITEM "C&opy\aCtrl+C", CM_EDITCOPY
    //MENUITEM "&Paste\aCtrl+V", CM_EDITPASTE
    //MENUITEM "&Delete\aDel", CM_EDITDELETE
    //MENUITEM "C&lear All\aCtrl+Del", CM_EDITCLEAR
```

```
  //END

  //POPUP "&Search"
  //BEGIN
    //MENUITEM "&Find...", CM_EDITFIND
    //MENUITEM "&Replace...", CM_EDITREPLACE
    //MENUITEM "&Next\aF3", CM_EDITFINDNEXT
  //END

  MENUITEM SEPARATOR

  MENUITEM SEPARATOR

  MENUITEM SEPARATOR

  MENUITEM SEPARATOR
END


// Menu merged in when TListView is active, notice the extra MENUITEM
SEPARATORs which are
// for menu negotation.  These separators are used as group markers by OWL.
//
IDM_LISTVIEW MENU LOADONCALL MOVEABLE PURE DISCARDABLE
BEGIN
  MENUITEM SEPARATOR

  //POPUP "&Edit"
  //BEGIN
    //MENUITEM "&Undo\aCtrl+Z", CM_EDITUNDO
    //MENUITEM SEPARATOR
    //MENUITEM "&Cut\aCtrl+X", CM_EDITCUT
    //MENUITEM "C&opy\aCtrl+C", CM_EDITCOPY
    //MENUITEM "&Paste\aCtrl+V", CM_EDITPASTE
    //MENUITEM "&Delete\aDel", CM_EDITDELETE
    //MENUITEM "&Add Item\aIns", CM_EDITADD
    //MENUITEM "&Edit Item\aEnter", CM_EDITEDIT
    //MENUITEM "C&lear All\aCtrl+Del", CM_EDITCLEAR
  //END

  MENUITEM SEPARATOR

  MENUITEM SEPARATOR

  MENUITEM SEPARATOR

  MENUITEM SEPARATOR
END


IDM_DOCMANAGERFILE MENU LOADONCALL MOVEABLE PURE DISCARDABLE
BEGIN
  MENUITEM "&New", CM_MDIFILENEW
  MENUITEM "&Open...", CM_MDIFILEOPEN
  MENUITEM "&Close", CM_FILECLOSE
  MENUITEM SEPARATOR
  MENUITEM "&Save", CM_FILESAVE, GRAYED
  MENUITEM "Save &As...", CM_FILESAVEAS, GRAYED
  MENUITEM SEPARATOR
  MENUITEM "Print Pre&view...", CM_FILEPRINTPREVIEW, GRAYED
  MENUITEM "&Print...", CM_FILEPRINT, GRAYED
  MENUITEM "P&rint Setup...", CM_FILEPRINTERSETUP, GRAYED
```

```
  MENUITEM SEPARATOR
  MENUITEM "E&xit\tAlt+F4", CM_EXIT
END


// Table of help hints displayed in the status bar.
//
STRINGTABLE
BEGIN
  -1,                   "File/document operations"
  CM_MDIFILENEW,        "Creates a new document"
  CM_MDIFILEOPEN,       "Opens an existing document"
  CM_VIEWCREATE,        "Creates a new view for this document"
  CM_FILEREVERT,        "Reverts changes to last document save"
  CM_FILECLOSE,         "Closes the active document"
  CM_FILESAVE,          "Saves the active document"
  CM_FILESAVEAS,        "Saves the active document with a new name"
  CM_FILEPRINT,         "Prints the active document"
  CM_FILEPRINTERSETUP,  "Sets print characteristics for the active document"
  CM_FILEPRINTPREVIEW,  "Displays full pages as read-only"
  CM_EXIT,              "Quits Graphing and prompts to save the documents"
  //CM_EDITUNDO-1,        "Edit operations"
  //CM_EDITUNDO,          "Reverses the last operation"
  //CM_EDITCUT,           "Cuts the selection and puts it on the Clipboard"
  //CM_EDITCOPY,         "Copies the selection and puts it on the Clipboard"
  //CM_EDITPASTE,          "Inserts the Clipboard contents at the insertion
point"
  //CM_EDITDELETE,        "Deletes the selection"
  //CM_EDITCLEAR,         "Clears the active document"
  //CM_EDITADD,           "Inserts a new line"
  //CM_EDITEDIT,          "Edits the current line"
  //CM_EDITFIND-1,        "Search/replace operations"
  //CM_EDITFIND,          "Finds the specified text"
  //CM_EDITREPLACE,       "Finds the specified text and changes it"
  //CM_EDITFINDNEXT,      "Finds the next match"
  CM_CASCADECHILDREN-1, "Window arrangement and selection"
  CM_CASCADECHILDREN,   "Cascades open windows"
  CM_TILECHILDREN,      "Tiles open windows"
  CM_ARRANGEICONS,      "Arranges iconic windows along bottom"
  CM_CLOSECHILDREN,     "Closes all open windows"
  CM_HELPABOUT-1,       "Access About"
  CM_HELPABOUT,         "About the Graphing application"
END


//
// OWL string table
//

// EditFile (include/owl/editfile.rc and include/owl/editsear.rc)
//
STRINGTABLE LOADONCALL MOVEABLE DISCARDABLE
BEGIN
  IDS_CANNOTFIND,       "Cannot find ""%s""."
  IDS_UNABLEREAD,       "Unable to read file %s from disk."
  IDS_UNABLEWRITE,      "Unable to write file %s to disk."
  IDS_FILECHANGED,      "The text in the %s file has changed.\n\nDo you want
to save the changes?"
  IDS_FILEFILTER,       "Text files|*.txt|AllFiles|*.*|"
END
```

```
// ListView (include/owl/listview.rc)
//
STRINGTABLE LOADONCALL MOVEABLE DISCARDABLE
BEGIN
  IDS_LISTNUM,  "Line number %d"
END


// Doc/View (include/owl/docview.rc)
//
STRINGTABLE LOADONCALL MOVEABLE DISCARDABLE
BEGIN
  IDS_DOCMANAGERFILE,   "&File"
  IDS_DOCLIST,          "--Document Type--"
  IDS_VIEWLIST,         "--View Type--"
  IDS_UNTITLED,         "Document"
  IDS_UNABLEOPEN,       "Unable to open document."
  IDS_UNABLECLOSE,      "Unable to close document."
  IDS_READERROR,        "Document read error."
  IDS_WRITEERROR,       "Document write error."
  IDS_DOCCHANGED,      "The document has been changed.\n\nDo you want to save
the changes?"
  IDS_NOTCHANGED,       "The document has not been changed."
  IDS_NODOCMANAGER,     "Document Manager not present."
  IDS_NOMEMORYFORVIEW,  "Insufficient memory for view."
  IDS_DUPLICATEDOC,     "Document already loaded."
END


// Printer (include/owl/printer.rc)
//
STRINGTABLE LOADONCALL MOVEABLE DISCARDABLE
BEGIN
  IDS_PRNON,            " on "
  IDS_PRNERRORTEMPLATE, "'%s' not printed. %s."
  IDS_PRNOUTOFMEMORY,   "Out of memory"
  IDS_PRNOUTOFDISK,     "Out of disk space"
  IDS_PRNCANCEL,        "Printing canceled"
  IDS_PRNMGRABORT,      "Printing aborted in Print Manager"
  IDS_PRNGENERROR,      "Error encountered during print"
  IDS_PRNERRORCAPTION,  "Print Error"
END


// Exception string resources (include/owl/except.rc)
//
STRINGTABLE LOADONCALL MOVEABLE DISCARDABLE
BEGIN
  IDS_OWLEXCEPTION,     "ObjectWindows Exception"
  IDS_UNHANDLEDXMSG,    "Unhandled Exception"
  IDS_OKTORESUME,       "OK to resume?"
  IDS_UNKNOWNEXCEPTION, "Unknown exception"

  IDS_UNKNOWNERROR,     "Unknown error"
  IDS_NOAPP,            "No application object"
  IDS_OUTOFMEMORY,      "Out of memory"
  IDS_INVALIDMODULE,    "Invalid module specified for window"
  IDS_INVALIDMAINWINDOW,"Invalid MainWindow"
  IDS_VBXLIBRARYFAIL,   "VBX Library init failure"

  IDS_INVALIDWINDOW,    "Invalid window %s"
  IDS_INVALIDCHILDWINDOW,"Invalid child window %s"
```

```
  IDS_INVALIDCLIENTWINDOW,"Invalid client window %s"

  IDS_CLASSREGISTERFAIL,"Class registration fail for window %s"
  IDS_CHILDREGISTERFAIL,"Child class registration fail for window %s"
  IDS_WINDOWCREATEFAIL, "Create fail for window %s"
  IDS_WINDOWEXECUTEFAIL,"Execute fail for window %s"
  IDS_CHILDCREATEFAIL,  "Child create fail for window %s"

  IDS_MENUFAILURE,      "Menu creation failure"
  IDS_VALIDATORSYNTAX,  "Validator syntax error"
  IDS_PRINTERERROR,     "Printer error"

 IDS_LAYOUTINCOMPLETE, "Incomplete layout constraints specified in window %s"
    IDS_LAYOUTBADRELWIN,    "Invalid  relative  window  specified  in  layout
constraint in window %s"

  IDS_GDIFAILURE,       "GDI failure"
  IDS_GDIALLOCFAIL,     "GDI allocate failure"
  IDS_GDICREATEFAIL,    "GDI creation failure"
  IDS_GDIRESLOADFAIL,   "GDI resource load failure"
  IDS_GDIFILEREADFAIL,  "GDI file read failure"
  IDS_GDIDELETEFAIL,    "GDI object %X delete failure"
  IDS_GDIDESTROYFAIL,   "GDI object %X destroy failure"
  IDS_INVALIDDIBHANDLE, "Invalid DIB handle %X"
END


// General Window's status bar messages. (include/owl/statusba.rc)
//
STRINGTABLE
BEGIN
  IDS_MODES             "EXT|CAPS|NUM|SCRL|OVR|REC"
  IDS_MODESOFF          "   |  |   |  |   |   "
  SC_SIZE,              "Changes the size of the window"
  SC_MOVE,              "Moves the window to another position"
  SC_MINIMIZE,          "Reduces the window to an icon"
  SC_MAXIMIZE,          "Enlarges the window to it maximum size"
  SC_RESTORE,           "Restores the window to its previous size"
  SC_CLOSE,             "Closes the window"
  SC_TASKLIST,          "Opens task list"
  SC_NEXTWINDOW,        "Switches to next window"
END


// Validator messages (include/owl/validate.rc)
//
STRINGTABLE LOADONCALL MOVEABLE DISCARDABLE
BEGIN
  IDS_VALPXPCONFORM     "Input does not conform to picture:\n""%s"""
  IDS_VALINVALIDCHAR    "Invalid character in input"
  IDS_VALNOTINRANGE     "Value is not in the range %ld to %ld."
  IDS_VALNOTINLIST      "Input is not in valid-list"
END


//
// Bitmaps used by the speedbar.  Each bitmap is associated with a
// particular menu command.
//
CM_MDIFILENEW BITMAP "new.bmp"
CM_MDIFILEOPEN BITMAP "pointfile.bmp"
CM_FILESAVE BITMAP "save.bmp"
```

```
//CM_EDITUNDO BITMAP "undo.bmp"
//CM_EDITCUT BITMAP "cut.bmp"
//CM_EDITCOPY BITMAP "copy.bmp"
//CM_EDITPASTE BITMAP "paste.bmp"

//CM_EDITFIND BITMAP "find.bmp"
//CM_EDITFINDNEXT BITMAP "findnext.bmp"

CM_FILEPRINTPREVIEW  BITMAP "preview.bmp"

CM_FILEPRINT BITMAP "print.bmp"


//
// Print Preview speed bar bitmaps
//
APX_PPR_PREVIOUS BITMAP "previous.bmp"
APX_PPR_NEXT BITMAP "next.bmp"
APX_PPR_ONEUP BITMAP "preview1.bmp"
APX_PPR_TWOUP BITMAP "preview2.bmp"
APX_PPR_DONE BITMAP "prexit.bmp"


//
// Misc application definitions
//

// MDI document ICON
//
IDI_DOC ICON "mdichild.ico"

// Application ICON
//
IDI_MDIAPPLICATION ICON "appldocv.ico"


// About box.
//
IDD_ABOUT DIALOG 12, 17, 204, 65
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "About 2D/3D TINs"
FONT 8, "MS Sans Serif"
BEGIN
  CTEXT "Version", IDC_VERSION, 2, 14, 200, 8, SS_NOPREFIX
  CTEXT "The 2D/3D TINs program", -1, 2, 4, 200, 8, SS_NOPREFIX
  CTEXT "", IDC_COPYRIGHT, 2, 27, 200, 17, SS_NOPREFIX
  RTEXT "", IDC_DEBUG, 136, 55, 66, 8, SS_NOPREFIX
  ICON IDI_MDIAPPLICATION, -1, 2, 2, 34, 34
  DEFPUSHBUTTON "OK", IDOK, 82, 48, 40, 14
END


// Printer abort box.
//
IDD_ABORTDIALOG DIALOG 84, 51, 130, 60
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Printing"
BEGIN
  PUSHBUTTON "Cancel", IDCANCEL, 46, 40, 40, 14, WS_TABSTOP
   CTEXT "Now printing Page %d of", ID_PAGE, 0, 8, 130, 8, SS_CENTER | NOT
WS_VISIBLE | WS_GROUP
  CTEXT "Now printing", -1, 0, 8, 130, 8,
```

```
  CTEXT "'%s' on the", ID_TITLE, 0, 16, 130, 8
  CTEXT "", ID_PORT, 0, 24, 130, 8, SS_CENTER | NOT WS_VISIBLE | WS_GROUP
  CTEXT "%s on %s", ID_DEVICE, 0, 24, 130, 8
END


// Version info.
//
#if !defined(__DEBUG_)

// Non-Debug VERSIONINFO
//
1 VERSIONINFO LOADONCALL MOVEABLE
FILEVERSION 1, 0, 0, 0
PRODUCTVERSION 1, 0, 0, 0
FILEFLAGSMASK 0
FILEFLAGS VS_FFI_FILEFLAGSMASK
#if defined(BI_PLAT_WIN32)
FILEOS VOS__WINDOWS32
#else
FILEOS VOS__WINDOWS16
#endif
FILETYPE VFT_APP
BEGIN
  BLOCK "StringFileInfo"
  BEGIN
     // Language type = U.S. English(0x0409) and Character Set = Windows,
Multilingual(0x04e4)
     BLOCK "040904E4"                // Matches VarFileInfo Translation hex
value.
    BEGIN
      VALUE "CompanyName", "\000"
      VALUE "FileDescription", "2D TIN & 3D TIN for Windows\000"
      VALUE "FileVersion", "1.0\000"
      VALUE "InternalName", "TIN\000"
      VALUE "LegalCopyright", "Copyright © 1999. All Rights Reserved.\000"
        VALUE "LegalTrademarks", "Windows(TM) is a trademark of Microsoft
Corporation\000"
      VALUE "OriginalFilename", "Graphing.exe\000"
      VALUE "ProductName", "2D/3D TIN\000"
      VALUE "ProductVersion", "1.0\000"
    END
  END

  BLOCK "VarFileInfo"
  BEGIN
    VALUE "Translation", 0x0409, 0x04e4    // U.S. English(0x0409) & Windows
Multilingual(0x04e4) 1252
  END
END

#else

// Debug VERSIONINFO
//
1 VERSIONINFO LOADONCALL MOVEABLE
FILEVERSION 1, 0, 0, 0
PRODUCTVERSION 1, 0, 0, 0
FILEFLAGSMASK   VS_FF_DEBUG   |   VS_FF_PRERELEASE   |   VS_FF_PATCHED   |
VS_FF_PRIVATEBUILD | VS_FF_SPECIALBUILD
FILEFLAGS VS_FFI_FILEFLAGSMASK
#if defined(BI_PLAT_WIN32)
```

```
FILEOS VOS__WINDOWS32
#else
FILEOS VOS__WINDOWS16
#endif
FILETYPE VFT_APP
BEGIN
  BLOCK "StringFileInfo"
  BEGIN
     // Language type = U.S. English(0x0409) and Character Set = Windows,
Multilingual(0x04e4)
     BLOCK "040904E4"                // Matches VarFileInfo Translation hex
value.
    BEGIN
      VALUE "CompanyName", "\000"
      VALUE "FileDescription", "2D/3D TIN for Windows\000"
      VALUE "FileVersion", "1.0\000"
      VALUE "InternalName", "2D/3D TIN\000"
      VALUE "LegalCopyright", "Copyright © 1999. All Rights Reserved.\000"
        VALUE "LegalTrademarks", "Windows(TM) is a trademark of Microsoft
Corporation\000"
      VALUE "OriginalFilename", "Graphing.exe\000"
      VALUE "ProductName", "2D/3D TIN\000"
      VALUE "ProductVersion", "1.0\000"
      VALUE "SpecialBuild", "Debug Version\000"
      VALUE "PrivateBuild", "Built by Alias Abdul-Rahman \000"
    END
  END

  BLOCK "VarFileInfo"
  BEGIN
    VALUE "Translation", 0x0409, 0x04e4    // U.S. English(0x0409) & Windows
Multilingual(0x04e4) 1252
  END
END

#endif
```